Decision Support System for Fighter Pilots

Randleff, Lars Rosenberg

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

Decision Support System for Fighter Pilots

Lars Rosenberg Randleff

Kongens Lyngby 2007
IMM-PHD-2007-172

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

During a mission over enemy territory a fighter aircraft may be engaged by ground based threats. The pilot can use different measures to avoid the aircraft from being detected by e.g. enemy radar systems. If the enemy detects the aircraft a missile may be fired to seek and destroy the aircraft. Such a missile will almost always be either radar guided or heat seeking. It will be launched from a permanent launch pad, or it will be man portable and small enough to fit in the boot of a car. The probability of a missile being detected by on-board sensors depends on the type of missile. If a missile is detected the pilot may choose to deploy electronic countermeasures to avoid the impact of the missile. The countermeasures to choose depend on e.g. the type of missile and guidance system, distance and direction between the missile and the aircraft, an assessment of the environment hostility, aircraft altitude and airspeed, and the availability of countermeasures.

Radar systems, guidance of missiles, and electronic countermeasures are all parts of the electronic warfare domain. A brief description of this domain is given. It contains an introduction to both systems working on-board the aircraft and countermeasures that can be applied to mitigate threats.

This work is concerned with methods for finding proper evasive actions when a fighter aircraft is engaged by ground based threats. To help the pilot in deciding on these actions a decision support system may be implemented. The environment in which such a system must work is described, as are some general requirements to the design of the system. Decisions suggested by the system are based on information acquired from different sources. The process of providing information from sources such as intelligence, on-board sensor systems, and

tactical data from other platforms (aircraft, ships, etc.) is described.

Different approaches to finding the combination of countermeasures and manoeuvres improving the survivability of the aircraft are investigated. During training a fighter pilot will learn a set of rules to follow when a threat occurs. For the pilot these rules will be formulated in natural language. An expert system can be build by translating these rules into a language understandable by a computer program. This is done in the development of a Prolog based decision support system.

A fighter aircraft decision support system is likely to base its decisions on input from non-perfect sources. Warnings from on-board sensor systems can be false and intelligence reports deficient. A Bayesian net is modelled to address this. Building the dependency tables of a Bayesian net requires a large number of cells to be filled with relevant probabilities. Not having sufficient knowledge about these probabilities makes the work with developing a Bayesian net cumbersome. Therefore a method for structural learning is investigated. Here a Bayesian net is build using a set of sample data from a number of missile flight simulations.

Knowledge about threats in the current combat scenario may influence the choice of evasive manoeuvres and proper countermeasures. If at any given time more expendable countermeasures are dispensed than necessary, and none is left for a later necessity, the survivability of the aircraft may decrease. A mathematical model is developed to describe this problem. It is solved to optimality using solver software. When new threats occur the decision support system must be able to provide suggestions within a fraction of a second. Since the time it takes to find an optimal solution to the mathematical model can not comply with this requirement solutions are sought using a metaheuristic.

Resumé

Når et jagerfly flyver over fjendtligt område, kan det blive udsat for jordbaserede trusler. For at undgå at blive opdaget af f.eks. fjendtlige radarsystemer, kan piloten benytte sig af forskellige modmidler. Er flyet først blivet opdaget, kan fjenden affyre missiler mod det. Sådan et missil vil næsten altid være enten radarstyret eller varmesøgende. Det kan blive affyret fra en permanent affyringsrampe, eller det kan være skulderbårent, og så lille at det kan skjules i bagagerummet på en bil. Sandsynligheden for, at sensorer ombord på flyet kan detektere missilet afhænger bl.a. af missilets type. Når et missil er blevet detekteret, kan piloten vælge at anvende modmidler for at undgå, at flyet bliver ramt af missilet. Hvilket modmiddel, der skal vælges afhænger bl.a. af missiltypen, hvordan missilet er styret, afstand og retning mellem missilet og flyet, en bedømmelse af, hvor fjendtlige omgivelserne er, flyets højde og hastighed og af hvilke modmidler der er tilgængelige.

Radarsystemer, styring af missiler og elektroniske modmidler hører alle til i domænet *elektronisk krigsførelse*. En kort beskrivelse af dette domæne er givet her. Beskrivelsen indeholder både en introduktion til systemer om bord på flyet, og en beskrivelse af de modmidler, som kan anvedes for at undgå missiler.

Arbejdet beskrevet i denne afhandling går ud på at finde ud af, hvad piloten skal foretage sig når flyet udsættes for jordbaserede trusler. I den forbindelse kan piloten benytte sig af et beslutningsstøttesystem, der kan være installeret i jagerflyets cockpit. Både den kontekst hvori et beslutningsstøttesystem skal fungere, samt generelle krav til designet af systemet er beskrevet. Systemets beslutninger vil være baseret på informationer fra forskellige kilder, og processen med at fremskaffe informationer fra efterretningskilder, sensorsystemer ombord på flyet og taktiske data fra andre andre fly, skibe, osv. er kort beskrevet.

Forskellige tilgangsvinkler til det at finde den optimale kombination af modmidler og manøvrer er blevet undersøgt. En del af det en jagerpilot lærer under sin uddannelse vil kunne sammenfattes i et sæt regler, som skal følges når jagerflyet møder en trussel. Disse regler kan formuleres i naturligt sprog. Ved at oversætte disse regler til et sprog, der kan forstås af et computerprogram, kan man udvikle et ekspertsystem. På denne måde er et beslutningsstøttesystem baseret på sproget Prolog blevet udviklet.

Et beslutningsstøttesystem kan basere sine beslutninger på data, der ofte vil komme fra fejlbehæftede kilder. Advarsler fra sensorsystemer ombord på flyet kan være fejlagtige, og efterretninger kan være mangelfulde. Et bayesiansk net er blevet udviklet for at kunne håndtere dette. Afhængighedstabellerne i et bayesiansk net skal udfyldes med et stort antal sandsynligheder. Det er besværligt at udvikle et bayesiansk net bliver, hvis der ikke på forhånd er tilstrækkelig kendskab til disse sandsynligheder. Derfor er en metode til automatisk generering af et bayesiansk net blevet undersøgt, og baseret på data fra et antal simulerede missilangreb er et net blevet konstrueret.

Valget af manøvrer og modmidler vil afhænge af tilgængelig viden om trusler i det aktuelle scenarie. Hvis der på et tidspunkt anvendes flere modmidler end nødvendigt, og der derfor ikke er nok modmidler tilbage, hvis de på et senere tidspunkt skulle blive nødvendige, kan dette øge flyets risiko for at blive skudt ned. En matematisk model er blevet udviklet for at beskrive dette. Et beslutningsstøttesystem skal kunne give forslag til forbedring af flyets overlevelseschancer i løbet af meget kort tid. Eftersom løsning af den matematiske model med en *solver* tilsyneladende ikke kan leve op til dette tidskrav, søges modellen løst ved brug af en *metaheuristik*.

Preface

This dissertation was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree.

Both concepts of electronic warfare and the need for a decision support system in fighter aircraft are described. Such a system must suggest actions to the fighter pilot that will increase his chances of surviving a mission when flying over enemy territory. For finding these actions four different technologies have been evaluated. Each of the technologies are described in the dissertation. The technologies are compared with regards to a number of requirements, and recommendations for further work within this area are made.

Acknowledgements

In November 2003 I began as a Ph.D. student at the Danish Defence Research Establishment (DDRE) with Per Husmann Rasmussen as my supervisor. Per had many ideas for the Ph.D. project and we spent days together discussing these. Sadly, Per became seriously ill, and he passed away in the Summer of 2004. While supervising this Ph.D. project for a short time only, large parts of the work on Bayesian Network (BN) described in this dissertation is still based on Per's ideas. During the work with the BN approach I visited Kristian G. Olesen at the Department of Computer Science at the University of Aalborg. Kristian evaluated the model developed and gave hints on improvements of both structure and running time.

At DDRE Gert Hvedstrup Jensen took over as my supervisor. Since Gert has an interest in the use of Prolog this was chosen as the next approach. Steen Søndergaard and Jim Titley took it upon them to introduce me to the frightening yet fascinating world of Electronic Warfare (EW). They willingly answered all of my more or less cryptic questions about missiles, guidance systems, and state of the art, and they enthusiastically reviewed all of my ideas.

Part of the work has taken place in the section for Operations Research (OR) at the department of Informatics and Mathematical Modelling at the Technical University of Denmark. Here I have had Professor Jens Clausen as my main supervisor. Inspired by OR courses taken, and under the advice of Jens, a mathematical model has been formulated. This has been done with assistance of Associate Professor Jesper Larsen.

In the summer of 2005 I had a two month stay at the Georgia Tech Research Institute (GTRI) in Atlanta, Georgia. Here I had a beneficial cooperation with Dr. Fred Wright on the formulation of time aspects in a combat mission. Here I also met Lee Simonetta who took time from his busy schedule to escort me to Tucson, Arizona, to Jacksonville, Florida, and to Marietta, Georgia. Randy Scott organized my stay at Georgia Tech, and he spent many hours showing me my way around the Georgia Tech campus and on sightseeing all over Atlanta.

All the people mentioned here have helped me in my work with the Ph.D project and with this dissertation, and I would like to thank them for their efforts. I would also like to thank all the people who spend time proof reading this dissertation. While they have corrected misspelled words, bad wording, and misinterpretations, the errors remaining are all mine. Thanks also to Henrik Jørgensen from Terma for supplying some of the pictures given in this dissertation.

Finally, I would like to thank my family. Both Ane and our son Christian have suffered from my regular absence, both physically and mentally, since the beginning of the project. Thanks also to our son Mads, who planned the date of his arrival so that I could return from Atlanta just before he was born.

Lyngby, March 2007

Lars Rosenberg Randleff

Acronyms and Abbreviations

AAM	Air-to-Air Missile
ACS	Aircraft Combat Survivability
AI	Artificial Intelligence
ANN	Artificial Neural Net
ATRIA	Automated Threat Response using Intelligent Agents
BN	Bayesian Network
BVR	Beyond Visual Range
CLP	Constraint Logic Programming
CMAT	Countermeasure Association Technique
CMOP	Countermeasure Optimisation Problem
DDRE	Danish Defence Research Establishment
DG	Decision Graph
DIRCM	Directional Infrared Countermeasures
DSS	Decision Support System
ECAP	Electronic Combat Adaptive Processor
ECCM	Electronic Counter Countermeasures
ECM	Electronic Countermeasures
EM	Estimation-Maximization
EO	Electro-Optical
EOB	Electronic Order of Battle
EPM	Electronic Protective Measures
ESM	Electronic Support Measures
EU	Expected Utility
EW	Electronic Warfare
EWMS	Electronic Warfare Management System

GAMS	General Algebraic Modeling System
GAPATS	General Aviation Pilot Advisory and Training System
GPS	Global Positioning System
GTRI	Georgia Tech Research Institute
HDD	Heads-Down Display
HUD	Heads-Up Display
ICD	Interface Control Document
IDAS	Integrated Defensive Aids System
IFF	Identification Friend or Foe System
INS	Inertial Navigation System
IP	Intermediate Point
IPB	Intelligence Preparation of Battlefield
IR	Infrared
JPD	Joint Probability Distribution
MANPADS	Man Portable Air Defence System
MAWS	Missile Approach Warning System
MCO	Missile Countermeasure Optimization
MFD	Multi-Function Display
MWS	Missile Warning System
OODA	Observe, Orient, Decide, Act
OR	Operations Research
PVI	Pilot-Vehicle Interface
RCS	Radar Cross Section
RF	Radar Frequency
RWR	Radar Warning Receiver
SA	Situational Awareness
SAM	Surface-to-Air Missile
SL	Structural Learning
TRP	Threat Response Processor
TTG	Time-to-Go
UAV	Unmanned Aerial Vehicle
UV	Ultraviolet

Contents

Chapter 1

Introduction

A fighter aircraft on duty will often fly over enemy territory as part of a mission. During this mission the aircraft may be engaged by enemy aircraft, or it may be the target of missiles fired from ground based launch pads. Over time more and more systems have been implemented aboard fighter aircraft in order to improve the pilot's awareness about the condition of the aircraft and the current situation in the world surrounding it. As the number and complexity of these systems increase, so does the quantity of threats to the aircraft. When new threats emerge, the pilot's means of mitigating these threats will change. Already known countermeasures may be applied in new and different ways, and new countermeasures are designed. When threats occur proper evasive actions often consist of combinations of manoeuvres and applied countermeasures. To determine the proper action, the pilot may benefit from a decision support system implemented on-board the aircraft.

1.1 Contents

In order to acknowledge the need for a decision support system on-board a fighter aircraft one has to understand the kind of threats an aircraft may meet, what type of information on-board sensors may provide to the pilot, and what he can do to avoid the threats. Most threats, and the relevant countermeasures,

either receive or emit electromagnetic radiation, and the domain is often referred to as Electronic Warfare (EW). This domain is described in Chapter 2. The intention with this chapter is to provide the reader with enough understanding about Electronic Warfare to understand the considerations given in designing a decision support system for fighter pilots.

In Chapter 3 the basics of a decision support system in the realm of electronic warfare are described. The context of such a system is described and requirements to the development of the system are specified. Already existing systems, and some academical approaches to designing them, are also described here.

The aim of the work documented in this dissertation is to explore a number of approaches to the development of a decision support system for fighter pilots. These approaches comprise Prolog (Chapter 4), Bayesian Networks (Chapter 5), formulating and solving a mathematical integer programming model (Chapter 6), and the use of metaheuristics to solve the mathematical model in due time (Chapter 7). These four approaches are compared in Chapter 8.

Throughout the dissertation the pilot of the aircraft will, for convenience, be referred to as he/him. The aircraft described is intended to be generic, and prices for missiles, countermeasures and aircraft, are all fictitious.

1.2 Readers Prerequisites

The intended reader of this thesis should have enough statistical literacy to comprehend the basics of Bayesian networks. To fully understand the chapters about mathematical modelling and metaheuristics, some degree of mathematical maturity is needed as well. To understand the brief introduction to logic and the Prolog programming language given, the reader will benefit from some experience with programming. Knowledge about fighter aircraft or electronic warfare is not needed, as these issues are covered sufficiently for the understanding of the approaches described. All sections containing mathematical theory are written without the use of lemmas, corollaries, and theorems. This is a deliberate choice to ease reading of these parts of the report. For the proper definitions, theorems, and proofs the reader is encouraged to consult the referenced textbooks.

Chapter 2

Electronic Warfare

A large part of the warfare involving fighter aircraft is based on the use of electromagnetic radiation. This type of warfare is referred to as Electronic Warfare (EW) (also known as *Electronic Combat*). EW is defined as military actions using electromagnetic radiation to estimate, use, reduce, or avoid enemy use of the electromagnetic spectrum.

The EW taxonomy can be divided into three main parts: Electronic Support Measures (ESM), Electronic Countermeasures (ECM), and Electronic Protective Measures (EPM). ESM is used to gain knowledge about the enemy using sensors based on electromagnetic radiation. To obstruct enemy use of ESM ECM is used. Finally EPM is used to lower the applicability of the enemy's use of ECM. Terms within these three classes of EW are described in this chapter. For more detailed descriptions on these subjects the books [41, 42, 46] are recommended. The threats, sensors, countermeasures, and fighter aircraft described in this and following chapters are all assumed generic.

2.1 The Electromagnetic Spectrum

Electromagnetic radiation is a common description of physical phenomena such as visible light, X-rays, radar, infrared, and ultraviolet radiation. All of these

describe physically variations within the electrical and magnetical fields. The variations are described as waves, with a wave characterized by either its wavelength (λ) or its frequency (f). With c being the speed of light, $c \approx 300,000$ km/s, the relation between λ and f is given by $\lambda = \frac{c}{f}$. In Figure 2.1 the wavelengths of parts of the electromagnetic spectrum are shown. A band in the electromagnetic spectrum is an interval of frequencies (or wavelengths). This section relates some bands with their role in EW.

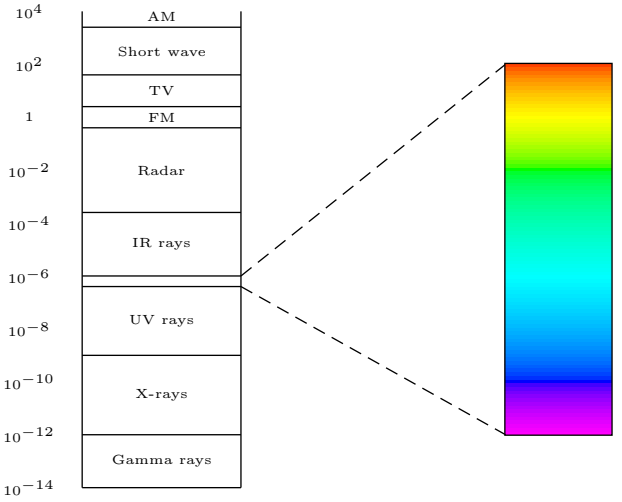


Figure 2.1: The electromagnetic spectrum, ranging from gamma rays to the wavelengths used for AM radio. The visual part of the spectrum is enhanced at the right.

Radar is an acronym for *RAdio Detection And Ranging*. Radar systems function by transmitting continually waves or short bursts of electromagnetic energy within the radar band, which can then be echoed off objects such as ships or aircraft. From the echo received by the radar system it is possible to determine the direction and range to the echoing objects. A Doppler radar calculates the velocity of an object using the difference in the frequencies between the emitted radar radiation and the radiation echoed off the object. Table 2.1 lists some radar technologies, their waveforms, and the parameters measured using them.

The radar band is itself divided into a number of sub-bands. For non-military use one set of names is used for these sub-bands, while another set of names is used within the EW domain. The sub-bands, their letter designations, and the wavelengths and frequencies dividing the subbands are shown in Figure 2.2.

Class:	Waveform:	Measures:
Pulse	Pulse	Range
Doppler	Continuous Wave (CW)	Velocity
	Pulse Doppler (PD)	Range and velocity

Table 2.1: Radar technologies in use. A Pulse radar system can be used to measure the distance to an object only, a radar based on Continues Wave technology will measure the velocity of the object, and a Pulse Doppler radar will find both the distance and the velocity.

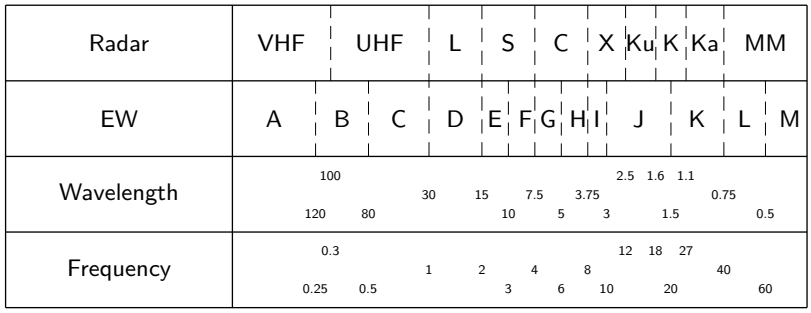


Figure 2.2: The radar sub-bands letter designations. The names in the top row refer to the ordinary radar sub-bands, while names in the second row refer to the names used in the EW domain. Wavelengths are given in cm and frequencies in GHz. See [41] for more details on the radar sub-bands.

A radar system can work in a number of modes, or independent radar systems working in different modes can work together in a single radar unit. The interception of an aircraft in the airspace covered by a ground-based radar is done by either a scanning radar or a multifunction radar in *scan* mode. When the aircraft is intercepted it may be tracked. When in *tracking* mode the radar will follow the aircraft to map its trajectory. Tracking the aircraft may lead to a missile being launched towards the aircraft. While the missile is approaching the aircraft the radar will be *locked* onto the aircraft. Since the energy and pattern of the radar radiation emitted in these different modes will also be different it is possible for radar receivers on-board the aircraft to distinguish between radar modes.

The amount of radar energy echoed from an object depends on the surface of the object facing the radar. The Radar Cross Section (RCS) of an object describes the reflection of an incident radar wave. It has the unit of a surface area which

should not be confused with the actual area of the object seen from the radar. The higher the RCS of an object the more power of an incident radar wave is echoed in the direction of the radar. Figure 2.3 shows the magnitude of the RCS for an aircraft as seen from different angles. It is found by measuring the radar reflection from angles around the aircraft.

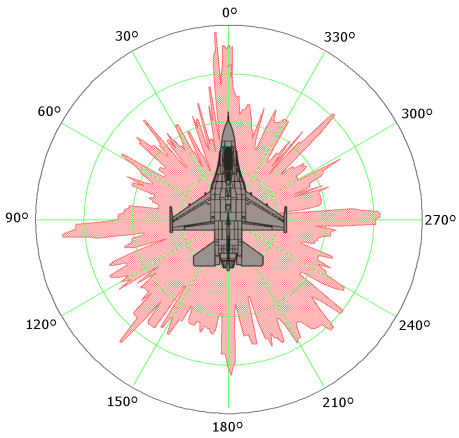


Figure 2.3: A polar plot showing the magnitude of the RCS of an aircraft measured at angles around the aircraft. (Polar plot is taken from [14]. Modifications made by the author.)

When in flight the friction from the surrounding air will heat up parts of the aircraft facing forward. Other parts may also have an increase in temperature caused by the engine exhaust plume. This heat results in the emission of electromagnetic radiation within the Infrared (IR) band. In Figure 2.4 the parts of an aircraft that will have an increased temperature have been marked. This radiation is used by *heat seeking* missiles, as described in Section 2.3.1.

2.2 Mission Scenarios

A fighter aircraft will typically be involved in one of two types of combat: air-to-ground combat where the enemy is positioned on the ground, and air-to-air combat where the aircraft is fighting other aircraft in mid-air. These two scenarios are described below.

Air-to-surface missions are often referred to as *raids* or *strikes*. According to [10]

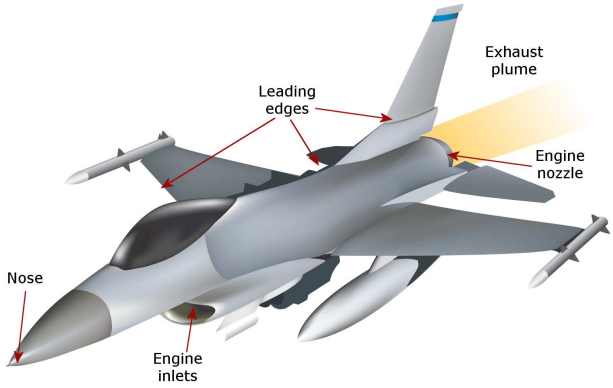


Figure 2.4: The parts of a fighter aircraft that will have an increase in temperature during flight.

"a strike is the delivery of a weapon or weapons against a specific target." The aim for the pilot in this scenario is to fly to a position near the target in high altitude, go to low altitude when approaching the target, deliver the weapon and return to high altitude before heading home. The reason for flying in and back at high altitude is to avoid ground based missile attacks. Flying above a given altitude will prevent attacks from both IR and Radar Frequency (RF) based missiles, while the aircraft will appear to be invisible to radar systems when flying below another altitude. The altitude profile of a strike is illustrated in Figure 2.5. The time it takes from descending the aircraft from high altitude to it is back at high altitude again will usually be a few minutes only. During these minutes the pilot has to focus on avoiding ground-based threats.

An aircraft is involved in a *dogfight* when it is fighting one or more enemy aircraft. When engaged in a dogfight the aircraft is manoeuvred to either avoid enemy missiles or bullets, or to attack an enemy aircraft with appropriate measures.

An enemy aircraft will often have the same mobility as the fighter pilot's own aircraft. When a fighter aircraft is engaged in a dogfight the threats can be positioned at any point in three dimensions and that will usually make the analysis of the current battlefield scenario more complex than for a single target mission.

Dogfights will usually be fought only as part of a *symmetrical warfare*. This

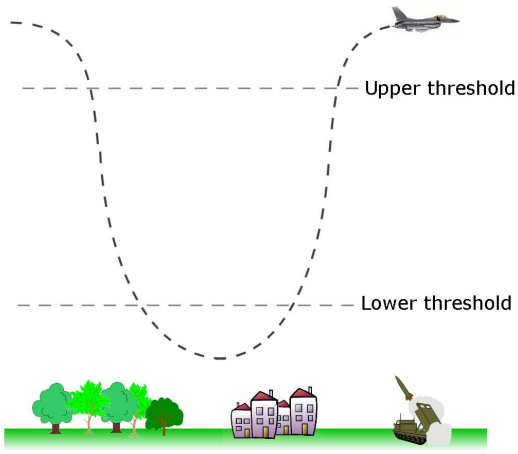


Figure 2.5: The altitude profile of a strike. The aircraft approach the target at high altitude, descend to deliver the weapon, and ascend to return home. The aircraft flies at a "safe" altitude when above the upper threshold or below the lower threshold.

means that the parts involved in the war have comparable forces, e.g. fighter aircraft. In *asymmetrical warfare* the forces of the one part are superior to the forces of the other part. Strikes may be part of both symmetrical and asymmetrical warfare.

2.3 Threats

To the fighter aircraft a ground based threat is either an enemy unit on the ground, capable of launching a missile towards the aircraft, or it is a launched missile itself. To the aircraft the best survivability is given if no missile is launched by the enemy. Flying at a "safe" altitude will make the aircraft less visible to the enemy, hopefully avoiding missiles being launched. If the threat is an enemy radar unit, flying the aircraft close to the ground will make the aircraft appear invisible due to ground clutter. If the enemy is capable of launching heat seeking missiles, the heat signature of the aircraft will be too small to lock onto if the aircraft is flying at a high altitude. When flying is required in a non-safe altitude the pilot may use pre-emptive measures such as a turning on the *jammer* or dispensing *flares* to prevent the enemy from locking on to the aircraft. Jammer and flares are examples of Electronic Countermeasures (ECM) which is

described in Section 2.5.

According to [6] about 650 different missile systems have been developed, and it is believed that 200-300 of these are still deployed[1]. A missile launched against an aircraft will be fired from either the surface of the earth (a Surface-to-Air Missile (SAM)) or from another aircraft (an Air-to-Air Missile (AAM)). Besides having a name given by the manufacturer many missile types are also given a USA/NATO type name, indicating the use of the missile. An example of this is the Russian *S-75 Dvina/Volkhov* that has the USA/NATO type name SA-2, indicating that it is a surface-to-air missile.

Some types of missiles are associated with one or more types of radar systems. Therefore the pilot may know which type of missile he is likely to encounter when knowledge about the type of a detected enemy radar system has been established. Knowing the missile type may give the pilot knowledge about how the missile can be countered. Since heat seeking missiles are not associated with a radar system, the pilot will not have this advantage when such a missile is launched.

For many types of missiles a direct hit at the aircraft is not necessary for it to have an impact. Many missiles are supplied with proximity fuses which will make the missile go off when it is within a certain range of the aircraft.

2.3.1 Guidance

Most missiles use some form of guidance in directing the missile towards the target. To avoid an incoming missile the pilot has to "break" the guidance (*break lock*), or transfer it from the aircraft to another object (*lock transfer*).

The guidance systems generally use electromagnetic radiation within one of two bands: Radar Frequency (RF) or Infrared (IR). If the missile is RF guided it is either equipped with a radar system of its own, or it is guided by a ground-based radar system. RF based missile guidance is *active* since it is based on emitting electromagnetic radiation to determine the position of the aircraft. When radar radiation is emitted from either the missile or from ground-based radar it may be detected by the aircraft, thus warning the pilot about an attack.

The IR based missile guidance is *passive* since it depends solely on radiation emitted from the aircraft and does not emit radiation itself. This type of missiles are equipped with an IR sensitive sensor that will guide the missile towards the

[1] As of February 2007.

aircraft. More sophisticated guidance systems are equipped with IR cameras that feed images to a *seeker algorithm*. This algorithm analyses the IR images to detect the aircraft and to distinguish it from false targets. The false targets may originate from objects in the scenario such as radiation from the sun or sparks from a welding unit, or they may be artificial targets created by the aircraft (see Section 2.5.3). A number of guidance systems are described in Appendix A.1.

As a rule of thumb the more energy that is emitted from or echoed off an object in the direction of a guidance system, the easier it is for the guidance system to follow the object. The pilot may manoeuvre the aircraft to reduce the amount of energy that is emitted towards an enemy observer. See Section 2.5.7 for a description of *breaklock zones*.

In many situations IR guided missiles, such as the Man Portable Air Defence System (MANPADS), will be the enemy's best choice of weapon. Often systems for launching these missiles are cheaper than systems using RF guidance, they are small enough to be stored in the boot of a car, they can be operated with little training, and until launched they are not easily detectable from the aircraft. Usually the missiles are launched when the distance to the aircraft is less than a few kilometres which will give the pilot only a few seconds to perform evasive actions. For these reasons IR guided missiles are often considered the greatest threat to both military and civilian aviation.

2.4 Electronic Support Measures

Equipment working within the electromagnetic spectrum to make the pilot aware of the combat situation surrounding the aircraft are known as Electronic Support Measures (ESM). The pilot bases his Situational Awareness (SA) on the ESM on-board the aircraft, and the better equipped the aircraft is, the better SA the pilot may obtain. In this section some of these measures are described.

2.4.1 Radar Warning Receiver

Different types of radar systems have different characteristics, and this is used by the Radar Warning Receiver (RWR) to determine from which type of radar system incoming radar waves originate. This is done by finding the properties of the wave in a lookup table. In this table the kind of missile often associated with the radar system may also be found. Based on the table a warning symbol

is shown in the *azimuth indicator*, and an audio warning is given to the pilot. The symbol displayed on the azimuth indicator shows the type of the radar system and the direction towards it. If the RWR can not show the type of radar system related to the detected radar signal, the azimuth indicator will indicate the radar system as being of an unknown type. Appendix A describes some of the RF-based threats detectable by a RWR. An azimuth indicator is shown in Figure 2.6.

Figure 2.6: An azimuth indicator as part of the Advanced Threat Display. (Photo courtesy of Terma.)

The position of a symbol shown in the azimuth indicator indicates the angle towards the threat and the proximity to the *lethal envelope* of the threat. The lethal envelope is the range in which the threat can engage the aircraft, and if the aircraft is close to, or within, the range of a threat this is shown in the azimuth indicator. For some azimuth indicators the symbols closest to the centre will represent the most imminent threats, while others will have these farthest away from the centre. While the first of these may seem most intuitive, the latter has its advantages. It will allow greater spatial separation of the highest priority threats on the display, making it easier for the pilot to determine directions to threats.

Usually the aircraft will be detected by enemy search radar before it is being tracked or locked upon. Radar characteristics vary from search radar to tracking radar and the RWR on-board the aircraft is able to distinguish between these radar modes based on the characteristics of incoming radar radiation. It is worth noting that not all symbols shown in the azimuth indicator represent threats. In any given scenario there may be numerous radars present, and possibly none or only a few of these represent a threat. Symbols representing search radars and acquisition and tracking radars may all be displayed simultaneously on the

azimuth indicator. Most newer RWR systems offer the possibility of prioritizing the threats and showing symbols for the threats with the highest priority only. Older RWR systems will only show the symbols of tracking radars and launched missiles.

2.4.2 Missile Warning System

The Missile Warning System (MWS) (sometimes referred to as Missile Approach Warning System (MAWS)) informs the pilot when a missile is approaching the aircraft. In a passive IR based MWS this is detected by continuously analysing IR images of the aircraft surroundings. These images are acquired using on-board IR sensors or IR cameras. If the images contain a hot spot (possibly indicating the plume of an approaching missile) that increases in size over a relatively short time span and which seems to follow the aircraft, a missile warning is issued.

In a passive Ultraviolet (UV) based MWS the images analysed are showing information from the UV part of the electromagnetic spectrum. This type of MWS has some benefits compared to the IR-based MWS since the UV characteristics of a missile plume may change during its flight. Information about the missile (time since launch, time to burn out, etc.) may then be extracted from the UV images.

A RF based MWS is an active system working in the radar band. It can determine the range and velocity of an approaching missile, thus giving the pilot an estimate of the time left before the aircraft is hit, known as the Time-to-Go (TTG). This helps the pilot to find the best point in time for performing evasive actions. A drawback to this kind of MWS is that missiles may be very hard to detect due to small RCS values. Another drawback is that missiles may be designed to follow the emitted radar radiation thus unfortunately converting the MWS into a missile attraction system.

2.4.3 Identification Friend or Foe

In a complex battle scene with many military platforms, including aircraft, ships, and/or ground-based vehicles, it might be difficult for the pilot to tell friend from foe. To help this the vehicles may be equipped with transponders identifying themselves. An aircraft equipped with an Identification Friend or Foe System (IFF) can then detect the transponder signal and it will identify the transponding vehicle as "friend" or "foe". Since not all aircraft are equipped with an IFF transponder, or a given transponder may not be turned on, the

pilot may not assume other aircraft not identifying themselves as "friends" to be, by default, "foes".

As with the RF based MWS a transponding IFF system will give away the position of the aircraft and it must be switched off if the presence of the aircraft is to be hidden from the enemy.

2.5 Electronic Countermeasures

The ESM onboard an aircraft continually informs the pilot about enemy threats. For the aircraft to counter these threats the aircraft may be equipped with a number of Electronic Countermeasures (ECM). These measures are used to either tell the pilot about the ESM used by the enemy, to disrupt the enemy's usage of his ESM, or a combination of both.

2.5.1 Jammer

A radar system will usually analyse the radar signals echoed off an aircraft. Depending on the type of radar system this analysis will decide the velocity, range, and/or direction to the aircraft. The results of this analysis might be used by the ground-based radars to determine when a missile must be launched against an aircraft. To confuse the analysis made by the radar system the aircraft can be equipped with a radar *jammer*. Different types of jammers exist: the simplest ones jams the radar signal by emitting a noise signal in the same frequency band as that of the radar signal. More advanced jammers calculate what radar signal to send out to make the results of the analysis in the receiving radar system erroneous, e.g. by estimating a wrong velocity, range, or angle. This may e.g. delude the radar into observing the aircraft as approaching while it may in fact be keeping its distance or even increasing it.

For a jammer to be effective against enemy radars the jamming signal needs to be emitted using more power than that of the echoed radar signal. Doing this will make the enemy radar interpret the actually echoed signal as noise compared to the jamming signal. The difference in power between the jammer signal and the echoed radiation is being used by some missile guidance systems. These will guide missiles towards any high-power signal, regardless of the information that may be found analysing this signal. A jammer that is turned on will thus serve as a beacon, possibly attracting the attention of an enemy radar operator. It it is therefore advisable to keep any jamming equipment turned off unless it is

considered necessary for the survival of the pilot to have it turned on.

2.5.2 Chaff

Chaff are small pieces of foil or bipolar material that immediately forms a cloud when dispensed from the aircraft. This cloud has a RCS comparable to that of the aircraft. This is used to make a radar system tracking the aircraft track the chaff cloud instead. The time it takes to form a chaff cloud is named the *bloom time*. After a few seconds the chaff cloud is dissipated and the aircraft will once again be visible to enemy radar. The process of forming a chaff cloud to decoy an approaching RF guided missile is shown in Figure 2.7.

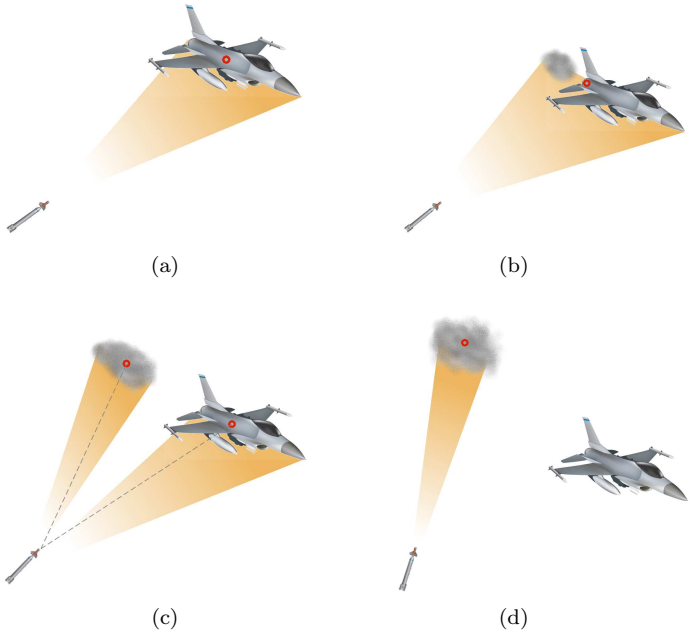


Figure 2.7: When chaff is dispensed it will form a cloud to decoy an approaching missile. In Figure 2.7(a) the centroid of the reflected radiation is positioned on the aircraft. Chaff is dispensed in Figure 2.7(b) and the centroid is moved backwards. In Figure 2.7(c) the missile has multiple targets to choose from, and in Figure 2.7(d) the chaff cloud has become the new target. With proper evasive manoeuvres of the aircraft the missile will not detect the presence of two targets and the lock will be directly transferred to the chaff cloud.

The tracking radar will follow an object within a given range gate only. If the aircraft can escape the range gate before the chaff cloud is dissipated it can not be tracked by the enemy radar before a new acquisition is performed.

The aircraft may be equipped with a number of chaff types and chaff dispensers. These will depend on a description of the battlefield and they are installed during the preparation of the aircraft. Chaff is an *expendable* countermeasure in that it can only be used for a limited amount of times before the inventory runs dry.

2.5.3 Flares

To escape from an IR guided missile the pilot may have to transfer the missile lock onto another object. This may be done by dispensing *flares*. Flares are another type of expendables which are made of hot burning material that forms an infrared signature which can be more attractive for the missile to follow than that of the aircraft. If the guidance system is designed to manoeuvre the missile towards the hottest spot within the visual range it might go for the flares instead of the aircraft. Although flares burn out within a few seconds this might be enough for the pilot to manoeuvre the aircraft away from the path of the missile.

When flares are dispensed they will soon get a speed remarkably smaller than that of the aircraft. The decrease in speed may be a signal to guided missiles that the object to follow (the aircraft) is not the object currently in focus (a flare). For flares to maintain the same speed as the aircraft they can be either tethered or self-propelled. Tethered flares are towed behind the aircraft at a fixed distance for a while, thus having the same speed as the aircraft itself. Propelled flares will start off by having the same speed as the aircraft. They will slowly decrease their speed, and the distance to the aircraft will increase.

If the pilot expects to be engaged by IR guided missiles, flares may be used pre-emptively. If numerous flares are dispensed before a missile is launched the missile may fail to acquire a lock on the aircraft itself.

As with chaff the number of flares and flare dispensers may vary according to a description of the battlefield and they will be set during the preparation of the aircraft as well. Flare dispensers may be directly linked to a MWS so a warning of an approaching missile immediately will trigger a flare dispense.

2.5.4 Directed Infrared Countermeasure

The Directional Infrared Countermeasures (DIRCM) is a system designed to protect the aircraft from IR guided missiles. When an approaching missile is detected by a MWS the DIRCM is directed towards the missile. When active the DIRCM uses pulses of IR energy to jam the IR seekers guiding the missile. The pulses of IR energy will generally have one of two effects: either the seeker is blinded and will loose focus on the aircraft long enough for the aircraft to break the lock, or it will mimic a thermal signature as that of the sun, thus forcing the seeker to look for alternative targets [1]. If the use of IR pulses is accompanied by the dispense of flares these may serve as new targets for the seeker and the lock is transferred.

2.5.5 Towed Decoy

As mentioned in Section 2.5.1 missiles may be guided toward an active jammer. To avoid this type of missiles while maintaining the effect of a jammer the jammer may be placed in a *towed decoy*. When deploying a towed decoy a wire connecting the decoy to the aircraft is unreeled and the decoy will be towed behind the aircraft at a fixed distance. When the towed decoy is no longer needed the wire may be re-reeled or simply severed.

The simplest towed decoys will have their own power supply and they will continue to jam for as long as the power permits. More sophisticated decoys may be connected to power supply and sensors on-board the aircraft. They will be able to adjust the jamming to the current battlefield scenario and they will continue to jam for as long as it is deemed necessary.

A towed decoy is kept at a safe distance behind the aircraft. At this distance an impact on the decoy by a missile will leave the aircraft undamaged. The aircraft manoeuvrability is limited when a towed decoy is deployed, so when it is no longer in use it must be severed or re-reeled. Before being deployed a towed decoy is usually placed under the aircraft fuselage or under either or both of the wings. This limits the total number of towed decoys to be deployed during a mission to one (the fuselage), two (both wings), or three (fuselage and wings).

2.5.6 Stealth

The highest survivability for the aircraft is obtained if it can fly by *stealth*, i.e. fly without being observed by the enemy. In designing a fighter aircraft several measures are taken to reduce the signatures of the aircraft to make it difficult for the enemy to observe. These measures include using radar absorbing materials and shaping the surface of the aircraft to obtain the smallest RCS values possible. A reduction of the IR signature of the aircraft is obtained by special designs of the airframe and propulsion system [10].

2.5.7 Breaklock Zones

The signatures of a fighter aircraft influence the success of an approaching missile. If the RCS of the aircraft is sufficiently small a RF guided missile will not be able to lock onto it. Likewise, an IR guided missile will have trouble following an aircraft that is almost invisible within the IR band. During flight the pilot will manoeuvre the aircraft to obtain the smallest signatures possible. An aircraft will typically have the largest RCS when seen from the side, while the RCS is often smallest when the aircraft is flying directly towards the radar receiver. To lower the IR signature of the aircraft the pilot may reduce the thrust and turn the aircraft so that hot surfaces are hidden by other parts of the aircraft.

The angles in which the aircraft has the lowest visibility to the enemy are known as *breaklock zones*. When a missile is locked onto the aircraft the pilot will manoeuvre the aircraft so that the enemy will become positioned within a breaklock zone. The manoeuvre will often be accompanied by the dispense of either chaff or flares, depending on the threat, so the lock can be transferred away from the aircraft.

2.5.8 Timing the Use of Countermeasures

When a threat is detected the use of appropriate countermeasures must be timed to gain the best possible protection. If applied too soon the countermeasure may have no effect, an applied too late the effect may not protect the aircraft. Dispensed too early a chaff cloud will be dissipated before having any effect on the missile, and the side-effect of having less chaff available will only decrease the aircraft's survivability at a later stage. If the chaff is dispensed too late the effect on the missile will not prevent it from reaching the aircraft. Similar considerations must be taken for IR guided missiles and flares. Here the time it

takes from launch until the missile reaches the aircraft is usually smaller than for RF guided missiles, and flares are usually dispensed as soon as the missile has been detected.

For on-board countermeasures such as jammer, towed decoy, or DIRCM, the time it takes before the countermeasure becomes active must be taken into consideration. While it may take only a few seconds for the jammer or the DIRCM to settle, or for the towed decoy to be unreeled, the use of these must be appropriately timed, just as for expendable countermeasures.

2.6 Electronic Protective Measures

ESM are used by the pilot to gain knowledge about the current battlefield scenario, and ECM are used for destroying the enemy's knowledge about the scenario. To spoil the use of ECM by an enemy aircraft one may use Electronic Protective Measures (EPM) (also known as Electronic Counter Countermeasures (ECCM)). While the descriptions here assume the user of EPM to be a, probably hostile, ground based radar system, EPM may also be applied in a fighter aircraft.

The fighter aircraft may have a RWR to determine the type of an enemy radar, or it may use either an on-board jammer or a towed decoy to deceive the radar. Some radar systems are designed to complicate the analysis done by either the RWR or the jammer. One technology for doing this is *frequency agility* where the radar system is able to shift the frequency in use. *Spread spectrum* technologies can be applied to prevent the aircraft RWR from correctly determine the kind of radar system in use. In spread spectrum the electromagnetic energy will be spread onto a large band within the radar spectrum. This will make the energy in each sub-band seem like background noise and it will be difficult for the RWR to recognize the radar signal.

2.7 The Fighter Aircraft

The fighter aircraft itself may limit the use of technology within the field of EW. These limits may be set by e.g. the design of the aircraft, the space available for additional EW equipment, or the manoeuvres required to gain maximum effect of countermeasures. Descriptions of these limits are given in this section.

Figure 2.8: Bombs and missiles mounted at stations underneath an F-16 fighter aircraft. (Photo courtesy of Erwin Stam.)

2.7.1 Adding Equipment

Newer models of fighter aircraft will be designed to comply with the demands raised by the use of EW equipment. This design is focused on e.g. lowering the signatures of the aircraft, and is one of the main issues covered in [10]. For existing aircraft new demands to EW equipment will lead to new configurations of the aircraft within the limits of the airframe given.

A typical fighter aircraft will have a number of stations for carrying bombs and missiles. These stations are placed underneath the wings and the fuselage of the aircraft, and the number of stations varies from one aircraft model to another. Since carrying missiles may enhance the RCS of the aircraft newer aircraft models are designed to carry missiles inside the body of the aircraft to maintain a low RCS. A fighter aircraft carrying bombs and missiles at its stations can be seen in Figure 2.8.

Adapting older fighter aircraft to carry new EW equipment can be a difficult task requiring structural changes to parts of the aircraft. To increases the EW performance of the aircraft with only minor structural changes some of the stations may be equipped with *pylons*. A pylon may carry e.g. the IR sensors for a MWS, a jammer, a DIRCM unit, or cartridges of chaff or flares. While a pylon takes up a station on the aircraft some pylons may function as stations themselves. Unfortunately pylons will often increase e.g. the RCS of the aircraft, and having them installed on the aircraft will thus not always improve the survivability of the aircraft.

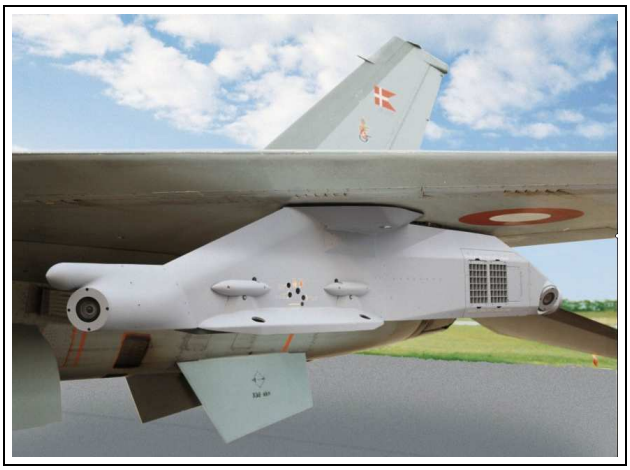


Figure 2.9: A pylon mounted under the wing. IR sensors are placed in the front and the back of the pylon. The holes on the rear end of the pylon will contain chaff or flare cartridges to be dispensed during flight. (Photo courtesy of Terma.)

2.7.2 The Cockpit

During combat the cockpit of a fighter aircraft constitutes a highly stressed environment. The pilot monitors a number of displays and indicating lights while listening to sounds of caution and danger. Besides this he has to maintain radio contact with allied aircraft and personnel placed on ships and ground while manoeuvring the aircraft at high speed.

Figure 2.10 shows the cockpit of the Falcon 4.0 flight simulator. While this is not a real-world aircraft the cockpit has high resemblance with the cockpit of a real F-16 fighter aircraft. The most important information about various avionics and aircraft systems is shown at the displays above the pilot's knees. The function of such a display, known as a Multi-Function Display (MFD), can be chosen according to the pilot's preferences. Above the left MFD the azimuth indicator shows the direction to enemy radars as found by the RWR.

As can be seen in Figure 2.10 there is limited space for adding controls and displays for new EW equipment. While extra functionality may be added to a MFD the pilot can only monitor a limited number of displays simultaneously. New equipment may add to the information available to the pilot, but it can not be allowed to add to the pilot's workload since this will only increase the probability of pilot errors.

Figure 2.10: The cockpit of an F-16 fighter aircraft. Above the knees of the pilot two MFDs can be seen. The azimuth indicator is positioned to the top left, above the left MFD. (Screenshot from the Falcon 4.0 flight simulator from Microprose.)

2.7.3 Manoeuvres

The effects of some countermeasures are increased if their deployment is accompanied by a swift aircraft manoeuvre. While the aircraft may have limited manoeuvre capability due to its design, the presence of a pilot in the aircraft will often limit the manoeuvres even more.

The acceleration caused by changing the direction of flight is often measured in g's, where one g equals the acceleration due to gravity. When the pilot is exposed to too much positive acceleration he may loose consciousness for a while. This is known as *black out* or *g-loc*, where *loc* stands for *loss of consciousness*. To prevent black out a fighter pilot may wear a *g-suit*. This suit applies pressure to the lower parts of the body to prevent blood from pooling. This will increase the amount of blood in the brain, hopefully keeping the pilot conscious. Negative acceleration may cause the pilot to experience *red out*, where capillaries in the eyes burst due to the increase in blood pressure. The bursting of capillaries may also cause haemorrhages in the brain, and like black out it can be lethal.

2.8 Summary

This chapter describes the domain of EW as the "battle of the electromagnetic spectrum". Threats may detect a fighter aircraft using radiation within one or more of the bands in the electromagnetic spectrum. Once the aircraft is detected a missile may be launched against it. This missile is most likely guided towards the aircraft using electromagnetic radiation. If the guidance system is passive it will rely on radiation emitted from the aircraft, e.g. IR radiation from hot parts of the airframe. A RF guided missile is an example of an active guidance system. It will emit radar radiation itself, and use the radiation echoed off the aircraft to determine the distance to, and possibly the velocity of, the aircraft.

The pilot may use ESM to gain knowledge about the current threat scenario. RWR and MWS are two such ESM systems. To counter threats the pilot may use different forms of ECM. ECM can either be equipment on-board the aircraft or it can be expendables dispensed into open air. When a missile is locked onto the aircraft the lock may be broken by appropriate use of ECM. The pilot may also use ECM preemptively to prevent threats from obtaining a lock on the aircraft. To reduce the effect of aircraft ECM the ground based threat may use EPM technologies. Using these may reduce the probability of the aircraft recognizing the threat or the threat being jammed by an aircraft jammer.

Chapter 3

Decision Support System in a Fighter Aircraft

On modern fighter aircraft more and more systems are implemented in order to improve the pilot's awareness about the current situation of the aircraft and the world surrounding it. As the number and complexity of these systems increase so does the quantity of threats to the aircraft and appropriate countermeasures for the pilot to choose from. To help the pilot decide on a proper evasive action when a threat occurs a Decision Support System (DSS) can be implemented aboard the aircraft [16, 20, 24].

This chapter describes the need for a DSS in a fighter aircraft, the requirements such a system must comply with, and the flow of data on which decisions from the system has to be made. Existing experimental and operational systems are also described.

3.1 Problem Description

In [37] a DSS is described as "a collection of computer-based interactive applications, which based on domain specific knowledge and information supports a decision maker in one or more phases of the decision process." A DSS may be

based upon an *expert system* which is a computer program that builds upon domain-specific knowledge from one or more experts. A DSS for fighter pilots will build upon expert knowledge within the field of EW.

Missiles may be fired from ground or sea based launch pads, or they may be fired from enemy fighter aircraft in an air-to-air engagement. In the latter case the workload on the pilot will be higher than it will be for land or sea based threats since the position, altitude, and speed of the enemy aircraft must be taken into consideration as well. In this work the subject is to investigate means to design a DSS for finding responses to ground based threats only.

When engaged by ground based threats one or more missiles may be launched towards the aircraft. The pilot may choose to use countermeasures to avoid the impact of an approaching missile. The countermeasures to choose depend on e.g. the type of missile, the distance and direction between the aircraft and the missile, the hostility of the environment, altitude of the aircraft, and the availability of countermeasures. Knowledge about threats that may be met in the near future may also influence the choice of proper countermeasures and the sequence in which they are used. If the aircraft is equipped with a limited amount of expendable countermeasures it might reduce the survivability of the aircraft for the entire mission if at any given time more expendables were used than necessary, thus leaving none for a later necessity.

Prototypes for the DSS are designed using techniques from the fields of Artificial Intelligence (AI) and Operations Research (OR). From the field of AI the possibility of using the Prolog programming language is examined. This is chosen since the tactics for responses to ground based threats can be formulated as a set of rules that can be implemented using Prolog.

At DDRE a Master's thesis has been written on the subject of using Bayesian Network (BN) for decision support for fighter pilots [16]. BN may also be considered as an AI technology, and it is chosen to expand on the experiences gained by that work by examining further use of BN.

The decisions suggested to the fighter pilot may improve if the DSS is designed to handle temporal aspects. These aspects may describe limits on the use of countermeasures during a mission. For this it is chosen to describe the problem using OR techniques. The problem is described by a mathematical model that can be solved to optimality. A metaheuristic is also applied, and here a trade-off between the quality of solutions and the time it takes to find them is made.

3.2 Survivability

The aim of this work is to design a DSS that may help to increase the survivability of the fighter aircraft when flying a mission over enemy territory. In [10] this survivability is named Aircraft Combat Survivability (ACS), and it is defined as: "The capability of an aircraft to avoid or withstand a man-made hostile environment." The survivability is related to the terms: *susceptibility*, *vulnerability*, and *killabillity* as described below:

Susceptibility. The susceptibility of an aircraft describes the inability to avoid missiles, radars, guns, and other elements of the hostile environment created by the enemy.

Vulnerability. When the elements of the hostile environment can not be avoided the vulnerability describes the inability of the aircraft to withstand the environment.

Killability. The killabillity describes the probability of the aircraft being "killed" due to enemy actions. This depends on both the susceptibility (the aircraft must be hit) and the vulnerability (this hit must cause sufficient damage to kill the aircraft) of the aircraft.

Survivability. The survivability is the opposite of the killability. Having a high probability of being killed will result in a low probability of surviving, and vice versa.

Throughout the literature on aircraft survivability the survivability is often referred to as P_S, while the killability is referred to as P_K [10]. The relation between P_S and P_K is $P_S = 1 - P_K$. For some threats the probability of having an impact on the aircraft can be established. If the probability for e.g. a proximity fused missile being fused by the aircraft is known as P_F and the probability of the aircraft being killed by a proximity fused missile is known as $P_{K|F}$ the survivability of such a missile attack is given by:

$$P_S = 1 - P_K = 1 - P_F \cdot P_{K|F}.$$

The probability $P_{K|F}$ depends on e.g. the construction of the aircraft. While P_S may be calculated for a given missile attack, finding it for an entire combat mission is more complex. Here the probabilities of e.g. the aircraft being detected by the enemy, and the enemy engaging in an attack of a detected aircraft must also be established.

3.3 Design Requirements

A DSS for fighter pilots must fulfil a number of requirements to be applicable during a mission. Below six of these requirements are described. In designing a method for suggesting actions to the pilot these requirements must be taken into consideration. In Chapter 8 the requirements are used in comparing four approaches for developing the DSS.

Real-time. The system has to find solutions to occurring threats in near real-time. It may take as little as 2-3 seconds from a missile has been launched until it reaches the aircraft. Before actions can be taken to avoid the incoming missile, sensors on-board the aircraft must detect the missile, the system must find an appropriate set of actions, and these actions need to be presented to the pilot. To give the pilot adequate time to perform evasive actions it is estimated that the system has approximately 200 milliseconds from a change in the threat scenario occurs until a set of actions has to be suggested to the pilot.

Hardware. Since a final implementation of the system must be run in a fighter aircraft, the hardware required for developing the system must match the requirements given to hardware in the aircraft. The results from a DSS depends on data input from other devices on-board the aircraft and hence it must be easy to integrate the DSS with these systems.

Updateable. The descriptions of threats and guidance systems are constantly evolving as are new countermeasure systems. Therefore, the system developed must be easily updated and maintained [24]. To ensure this the system must preferably be data driven, and updating the system will then be a matter of updating data on missile systems, guidance methods, etc. The algorithms used must have none or minor updates only.

Trustworthy. Any solution from the system must seem reasonable to the pilot. Otherwise the system will not be trusted and hence not used. This requirement can be divided into two sub-requirements: the system must suggest a reasonably solution to any changes in the scenario, e.g. when a new threat occurs, and when no threats are imminent no suggestions should be given.

Both in combat and during tests in the development phase the user of the system will be a fighter pilot. If the pilot does not understand how the solutions suggested by the system can be inferred he may not trust their validity. Therefore the concept of the inferring parts of the system must be relatively easy to understand.

Useful. The usefulness of a DSS is its ability to improve the survivability of the pilot. If the system is limited to only suggest actions to the pilot within a fraction of all the situations he may find himself and the aircraft in it is of no use. The difference between this requirement and the previous is that a system may be trustworthy only within given limits without being useful to the pilot. An example hereof is a system that suggests actions to mitigate e.g. RF threats only.

User Interface. Results from the system must be presented to the pilot in such a way that they are easy and fast to interpret. The presentation can be visual, use audio, or be a combination of both. During the evaluation of possible techniques for developing the system the presentation of the suggested action is of minor importance. In the development of a system that is to be fielded this requirement must take a high priority.

In [24] more issues are mentioned as critical to the design of a DSS. Among these issues are the use of data compression techniques, a user friendly database interface for updating the data used by the system, and effective memory management. None of these issues are considered crucial in this work and the handling of these is beyond the scope of the work.

Data from different sensors, or from the same sensor over a period of time, may be fused to give the pilot a better situational awareness. The discipline of performing data fusion is a topic of its own, and it too is considered beyond the scope of the work.

3.4 Mission Data Flow

Before a mission is initiated information about possible positions of enemy radars and launch pads are collected. This is done during a phase of the preparations known as the Intelligence Preparation of Battlefield (IPB). From this the Electronic Order of Battle (EOB) is established. The EOB describes the battlefield in details used by the pilot and by different systems on-board the aircraft. During the mission the pilot and on-board systems will continually retrieve information about the battlefield from on-board sensors and possibly also from other sources, such as ground personnel or other aircraft. After the mission the fighter pilot will be debriefed, and any observations during the mission will be recorded for later use. The data flow shown in Figure 3.1 describes the collection of data before, during, and after a mission is flown

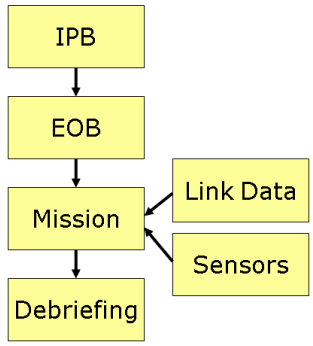


Figure 3.1: Mission data flow.

3.4.1 Intelligence Preparation of Battlefield

The IPB is described as a military method for collecting and processing intelligence about the battlefield [30]. The method describes how accurate and relevant intelligence may be organized and provided to a military decision maker in a timely fashion.

Sources for intelligence may vary from eyewitness statements and reports from military personnel to radio intercepts and satellite photos. The IPB describes how intelligence is to be used to gain knowledge about the battlefield. It is a repeated process consisting of the four steps described below:

1. Define the battlefield. The battlefield is a geographical area and the airspace above it. The decision maker will concentrate on decisions regarding forces within this area. The size of the area may vary during combat according to the knowledge of the battlefield available.
2. Describe the effects of the battlefield. Here the influence on the mission by e.g. the terrain or the current weather is established. This step may find corridors in where the aircraft can fly without being detected by enemy radar, or routes in where the aircraft can remain almost invisible due to fog.
3. Evaluate the threat. A profile of the enemy's capabilities within the battlefield is created. This may e.g. include the number and positions of enemy radar system.
4. Determine enemy courses of action. With the knowledge gained from the first three steps of the IPB the probable courses of enemy actions are determined.

3.4.2 Electronic Order of Battle

The EOB is defined as a list of the locations, identifications, functions, and capabilities of electronic equipment employed by a military force [41]. This information is made available to the pilot during the pre-flight preparation of the aircraft. Information about the planned route and current equipment and ammunitions on the aircraft may also be loaded electronically to an aircraft computer. The planned route will often be described using a fixed number of locations. Such a location is known as an Intermediate Point (IP).

The inventory consisting of countermeasures and weapons is based on the EOB. Since countermeasures and weapons will often have to share the same stations placed under the aircraft the more countermeasures that are deemed necessary the fewer weapons may be carried. Generally the assessment of the battlefield will lie within one of the categories given below. While battlefields within some of these categories are estimated as being unlikely, they are mentioned here for completeness.

- The battlefield contains no known threats and the aircraft is not equipped with ECM.
- The enemy has passive missiles only (e.g. MANPADS) and the aircraft will be equipped with flares.
- The enemy has active missiles only and the aircraft will be equipped with chaff. This situation is unlikely.
- The enemy has active missiles only. These missiles may be jammed and the aircraft is equipped with both chaff and jammer. This situation is unlikely as well.
- The enemy has multiple types of weapon or the composition of weapons is unknown. The aircraft is equipped with chaff, flares, and a jammer.
- The enemy has passive missiles only or missiles that may be jammed. The aircraft is equipped with jammer and flares. This situation too is unlikely.

3.5 System Data Flow

The basic data flow for the working environment of the DSS is shown in Figure 3.2. At the left hand side data is fed to the DSS from different systems on-board the aircraft. With the aid of a knowledge base and the constructed SA the

DSS finds a solution. This solution can then be presented to the pilot, and/or automatic responses from other on-board systems can be initiated.

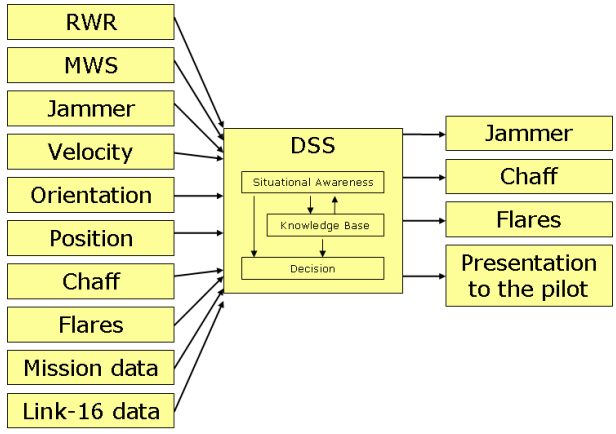


Figure 3.2: Data flow. Data is coming from sensors on the left hand side of the figure. Decisions from the DSS influence the subsystems on the right hand side.

The pilot is continuously gathering information from the on-board systems to improve and revise his Situational Awareness (SA). In order to support the pilot the DSS needs to build and maintain its own SA. When a solution is found by the DSS it must be executed. One way to do this is to present the solution to the pilot and let him execute the actions suggested. Since one of the reasons for introducing a DSS is to reduce the workload of the pilot the DSS may be linked directly to relevant subsystems for automatically deploying countermeasures, possibly in conjunction with proper evasive manoeuvres.

Decision making in military domains are often described by the four actions: Observe, Orient, Decide, Act (OODA). These actions are performed repeatedly in what is known as the OODA loop [5, 37]. When engaged by a missile the fighter pilot will first observe the missile; he will determine if the missile is posing an immediate threat; if it is he will decide on proper evasive actions; and finally he will act to avoid the impact of the missile. If the fighter pilot is to be aided by a DSS it will itself run through the first three phases of the OODA loop before a decision is presented to the pilot. If a proper response must be found no later than 200 milliseconds after a threat occurs, according to the requirements mentioned in Section 3.3, the DSS must perform a loop at least five times a second.

3.5.1 Acquiring Data

In order to decide on actions to suggest to the pilot the system needs input from various sources. In most fighter aircraft these sources will be systems connected to a data bus on-board the aircraft. In many aircraft this bus will comply with the MIL-STD-1553B standard [4].

On a MIL-STD-1553B bus the communication is controlled by a *bus controller*. Data can come from a number of *remote terminals*, and it can be read by a number of *bus monitors*. The bus controller serves as an arbiter that allows the remote terminals to use the bus one at a time. Data on the bus is transmitted as datagrams from one remote terminal to one or more bus monitors. The format of a datagram is described in an Interface Control Document (ICD), and since every remote terminal may use a specific format for every bus monitor receiving data from the terminal, a large number of ICDs may be needed to describe communication on the bus. Since a DSS may need input from many remote terminals, and these may vary from one aircraft to another, the DSS must be designed so it can be easily adapted to a comply with new sets of ICDs.

Most datagrams are transmitted with a relatively low frequency. If allowed by the bus controller a typical datagram will have a transmission frequency in the order of 1 to 20 per second, depending on the assessed importance of the datagram. Combined with a relatively low clock frequency controlling the bus, a slack time in the magnitude of 0.1 to 0.5 seconds may well appear between the time a sensor system has detected a threat till a bus monitor (e.g. the DSS) receives notification about it. This slack leaves only a short period of time for the DSS to find and suggest an action to the pilot.

Besides on-board sensor systems data to the DSS may be supplied through a tactical data link. Link-16 is a standard for such a tactical data link, and it is used for sharing information (e.g. identification and voice commands) between allied units (aircraft, ships, etc.) in the battlefield. A DSS may benefit from data given through Link-16 to maintain a model of threats, their tracks, sizes, numbers, and positions.

The sensors and sources for detecting threats on-board the aircraft differ in several aspects. They operate in different bands of the electromagnetic spectrum, use different methods to determine the threats, and since they may interpret their analogue input differently, they may not agree on the threats found. To give the DSS a single "ground truth" to work with, it may be essential that data from the different kind of sources are fused before handed to the system.

3.5.2 Threat Evaluation

Not all changes to the threat scenario will require the DSS to suggest new actions. Throughout a mission one of the goals of the DSS is to minimize the workload of the pilot. Therefore, when the aircraft flies over friendly territory, and no threats are imminent, no actions must be suggested by the DSS. As soon as the MWS issues a warning, or the RWR detects radiation from a possibly hostile radar, the territory can no longer be considered friendly, and the system will suggest actions to the pilot.

Even while flying over enemy territory the DSS must only suggest evasive actions when necessary. To determine when e.g. warnings from the RWR stem from previously undetected radar system, or if the radar has just reappeared on the RWR after being lost for a moment, a *threat evaluator* subsystem may be added to the DSS. Such a subsystem will have to base its evaluation on a registration of recent warnings. The RWR may loose track of radar systems if the aircraft is positioned such that either itself or the terrain underneath it prevents the radar from being detectable by the RWR antennas. For the MWS some warnings can be ignored too. If a warning is issued for a fraction of a second only, it is likely to be caused by e.g. a reflection that is visible only for a moment, instead of by a missile following the aircraft. If the threat evaluator has to compare every warning to previous warnings, the first warnings of a threat will be ignored. When warnings are no longer ignored even less time is left for the DSS to find a solution.

3.5.3 Executing Decisions

In general a DSS will support a decision maker in deciding proper actions. When the suggestions from a fighter aircraft DSS is presented to the pilot, the pilot will have to decide on the actions to perform before carrying them out. Since a limited time is available for the pilot to do this the DSS can perform the actions itself; even without the consent of the pilot. Most fighter aircraft use fly-by-wire control, i.e. all controls are entirely electronic. This means that the DSS can take control of the aircraft and perform the proper manoeuvres while dispensing expendables. While this may be technical feasible it is not necessarily a situation wanted by the pilot. According to [39] nine of ten interviewed pilots said that having the aircraft taking control was anathema to them.

Having the DSS working in interaction with some of the ECM system on-board the aircraft, without taking complete control, may still improve the survivability of the aircraft. The DSS may e.g. notify the jammer about RF threats not detected

by the jammer itself, or it may release flares or chaff when the aircraft is flying in the proper direction. While the control of the aircraft remains in the hands of the pilot, the DSS will increase his survivability and part of his workload will be taken from him.

3.6 Models and Systems

The approaches to the development of a fighter aircraft DSS described in this work are not the first attempts on introducing a DSS into the field of aviation. This section describes a number of experimental models that may be used in a DSS, and a few existing systems. Some of the models use techniques from the fields of AI and OR, such as *fuzzy sets*, *mimic nets*, and *genetic programming*. These techniques are not described here.

3.6.1 Experimental Models

The Countermeasure Association Technique (CMAT) automatically recommends countermeasures and manoeuvres to a pilot under missile attack [24]. The technique selects responses that will increase the survivability in relation to both the current and future threats. It is done by fusing possibly incomplete sensor input and feeding the results to a mimic net. A set of combat scenarios is presented to EW experts and the mimic net is trained to mimic the responses from these experts. Solutions found by the CMAT are called *strategies*, and given a threat scenario, a number of strategies are found. The best of these strategies, i.e. the strategy that best mimics the responses from EW experts, is chosen.

In 1986 the *Pilot's Associate* project was initiated by the United States Defense Advanced Research Projects Agency [11, 39, 48, 49]. The concept of the Pilot's Associate is a set of knowledge-based subsystems: two assessors, two planning subsystem, and a Pilot-Vehicle Interface (PVI). The Situation Assessment subsystem determines the current state of the outside world, while the System Status subsystem acquires knowledge about the state of on-board systems. The Tactics Planner subsystem supports the pilot in choosing responses from possible alternatives when immediate threats occur. In case of deviations from the mission plan the Mission Planner will adapt the plan to the current situation. The PVI adapts the information shown to the pilot to the current situation. When for instance the pilot needs to respond to an imminent threat, warnings that can be delayed, e.g. of malfunctioning subsystems, will not be shown.

In [48, 49] the Pilot's Associate is compared to a French system named *Copilote Electronique*. This system is based on an in-depth analysis of the pilot's cognitive processing, and the aim is to find ways to support the pilot based on his current activities and an estimation of his current mental workload.

The concept of General Aviation Pilot Advisory and Training System (GAPATS) is to run a flight management system on an inexpensive PC [33]. It is inspired by the Pilot's Associate system, and it consists of two main parts, the *Flight Mode Interpreter* (FMI), and the *Pilot Advisor* (PA). Based on e.g. aircraft state variables, such as altitude, airspeed, and rate of climb, the FMI determines the current flight mode (taxi, take off, cruise, etc.). This is done using fuzzy sets and the result is continually fed to the PA. Based on the current flight mode and a set of rules the PA will use the Heads-Up Display (HUD) and the Heads-Down Display (HDD) to display relevant information to the pilot.

The Missile Countermeasure Optimization (MCO) problem is the subject of a number of papers, e.g. [31, 32]. In [31] the aim is to combine aircraft manoeuvres and countermeasures to survive an attack from a single surface-launched anti-aircraft missile. The best combination of countermeasures and manoeuvres is found using genetic programming. The method encompasses uncertainties about both the type and the state of the approaching missile.

The Automated Threat Response using Intelligent Agents (ATRIA) system is described in [36]. Here Prolog is used in implementing *asset agents*. An asset agent represents a military asset such as a missile and/or aircraft tracking system, a radar system, or a missile. The agents in the system share an evolving description of the combat scenario, and while finding responses for themselves they communicate these to other friendly assets in the battlefield.

In [47] the use of temporal reasoning in the process of decision-making is described. Here the scenario is Beyond Visual Range (BVR) combat with multiple enemy aircraft, and the scope is to give the pilot a visual interpretation of the situation. For each of the enemy aircraft a *goodness value* is calculated, indicating which aircraft will have the upside in a close encounter. The goodness value is based on the time it will take for each aircraft to initiate appropriate actions. BNs are suggested as a means of evaluating the properties of an enemy fighter aircraft.

In [19] chaff and flare programs are optimised using a statistical model. This model express the survivability as a function of various parameters including missile attack rate, false alarm rate, missile detection probability, and the duration of a mission. The goal of the work described is not to optimise the survivability in a specific scenario, but rather to study how different parameters affect the chaff and flare programs.

3.6.2 Existing Systems

An Integrated Defensive Aids System (IDAS) is a system that collects input from different sources, and combines these to e.g. automatically dispense expendables or utilize a jammer to counter enemy systems. Input may come from different sensors on-board the aircraft, or it may be relayed from sensors on other platforms such as other aircraft, or ground- or sea-based sensor systems. Historically EW equipment on an aircraft has been considered as add-on units and not as integrated parts of the aircraft. In [52] it is mentioned that an IDAS must be considered as an integrated part of the aircraft systems in much the same way as the engine, and it is argued that the role of both is to get the aircraft to the target and back. Each of the two systems described in this section can be regarded as an IDAS.

The Electronic Warfare Management System (EWMS) is developed by Terma to reduces the pilot's workload and to ensure coordinated and effective use of on-board EW systems [51]. It can be operated in three modes: In manual mode the pilot activates the countermeasure program to use. When in semi-automatic mode the Electronic Combat Adaptive Processor (ECAP) will analyze the presence of threats and chose an effective combination of countermeasures. This combination is presented to the pilot who can give his consent by activating a switch. After consent is given a script for the combination of countermeasures is initiated. The pilot does not need to give his consent when the system is in automatic mode. A script related to the best use of countermeasures found will be initiated.

The Threat Response Processor (TRP) is developed at Georgia Tech Research Institute (GTRI) [55]. It is a combination of straightforward improvements of the survivability and an expert system, and it has undergone extensive flight testing over more than seven years. At first inputs from various systems on-board the aircraft are correlated, ambiguities are handled, and inputs describing e.g. different threats are prioritized. After this a number of scripts are spawned, depending on the character of the input. Among other things these scripts select proper countermeasures to apply and messages to display to the pilot. After each spawned script is executed the optimal response is found from the results of the scripts. This response is then fed to the EWMS to handle e.g. dispensing of expendables.

3.7 Summary

This chapter describes the need for a fighter aircraft DSS to assist the pilot in finding the best responses when being engaged by ground-based threats. Since solutions from a DSS are intended to improve the survivability of the pilot, the concept of survivability is introduced. The survivability depends on the susceptibility, vulnerability, and killabillity of the aircraft.

A DSS must comply with a number of design requirements. Requirements to response time, ability to run on aircraft hardware, ability to be easily updated, and the system being both trustworthy and useful are described. In addition to this a DSS must have an easy to use user interface/PVI. The design of the user interface is outside the scope of the work.

A DSS will work on mission data in determining optimal responses. The mission data flow is described, and the concepts of Intelligence Preparation of Battlefield (IPB) and Electronic Order of Battle (EOB) are introduced. When airborne the DSS relies on data from on-board sources. Data may often be acquired via an aircraft data bus. MIL-STD-1553B is a standard for such a bus, and some of the characteristics of this standard are mentioned.

Experimental models and existing systems are described. The experimental models apply techniques from the fields of AI and OR while systems that has already been implemented in aircraft seem to use more pragmatic approaches.

Chapter 4

The Prolog Approach

Prolog is a programming language for describing relations using logic. The name Prolog is the short concatenation of *PROgramming* and *LOGic*. In constructing a DSS Prolog can be applied for the formulation of a set of rules for the responses to ground-based threats and launched missiles. This chapter describes how to use the Prolog programming language for analysing a threat scenario, and for the construction of a Prolog based DSS.

Starting with some logic theory the basics of Prolog theory is introduced. Some textbooks on AI describe the terminology used in logic, and the use of logic in inferencing. See [54] for more details on logic, and [12, 56] for introductions to Prolog. The introductions of logic and Prolog in this chapter aim at providing the background for understanding the Prolog based DSS developed. This DSS is introduced in Section 4.5.

4.1 Motivation

When a fighter pilot is taught the essentials about reacting to ground-based threats, he is essentially given a set of rules to follow. These rules can be formulated in natural language by experts in the domain of EW. Rules such as

"if you get a missile warning, and not a radar warning, a missile using infrared guidance is launched" or "if you keep the altitude low enough, you can escape radar based threats" can be reformulated in Prolog, and used in the foundation of a Prolog-based DSS.

The set of rules written in Prolog is referred to as a Prolog program, and to use it for decision support, it should be fed to a computer program for execution. In this work the program executing the Prolog program is referred to as the Prolog *interpreter*, although Prolog programs can also be executed as compiled code. Since Prolog interpreters exist on many computer platforms, and input such as sensor responses and mission descriptions can easily be given using Prolog, the development of a Prolog based DSS does not require any special system setup.

Although many Prolog interpreters have features that enhance the strength of Prolog, "pure" Prolog is a relatively simple programming language. Therefore implementing a Prolog interpreter on a computer platform usable in a fighter aircraft is a feasible task, and transferring a desktop version of a DSS to an aircraft can be expected to be smoothly done.

It is estimated that writing a program in Prolog will often require far less development time than writing an equivalent program in e.g. C or C++ [12]. The syntax of Prolog programs makes them relatively easy to read, even for people with little or no programming experience. This helps in the validation of a DSS, since this can then be done partly by domain experts. If care is taken during development, and the program is written in an almost self-explanatory way, the task of maintaining a Prolog based DSS can relatively easy be done by people other than the original developers.

Even if Prolog may not be the preferred technique to use for a DSS, a Prolog based program may be used to test a DSS based on other techniques. If the answers from the Prolog system have been validated, it can be used as a look-up table for the "correct" responses in the scenarios it is used to evaluate.

4.2 Basic Theory

This section presents some basic theory about logic and Prolog. The vocabularies on both logic and Prolog are relatively large, and some of the basic terms are introduced here.

In essence the interactions in a domain, and data describing it, are given in Prolog using a set of *rules*. These rules are described in one or more files, which

can be interpreted by a Prolog interpreter. When doing so, the interpreter can answer questions about the domain, in accordance to the rules given. Prolog assumes a *closed world*, meaning that anything not explicitly declared is per definition non-existing.

Some Prolog systems offer graphical user interfaces, database connectivity, special constructs for software-based agents, and other features that enhance the usability of the system. For the evaluation of Prolog as the base of a DSS, such constructs are not necessary and they will not be described here.

Programming languages such as C/C++ are classified as *imperative*, which means that programs written using these languages describes the steps necessary for a program to evaluate input and produce output. These steps are described using statements, and the order of execution of these statements is determined by the programmer. As opposed to the imperative languages, Prolog is a *declarative* programming language. Here the emphasis is on declaring the relations between input and output, and not on how these relations should be implemented by the programmer.

4.2.1 Logic

Logic is a mathematical discipline working with statements that can be either *true* or *false*. This is done using *predicates*, which are functions mapping their arguments into true/false. A predicate with arguments, or a negation of the predicate, is known as a *literal*. When using logic in describing a domain, the description may include *objects*. *Variables* ranging over the objects of the domain can also be included.

Treating weather as an object, it can be described using a number of variables such as precipitation in millimetres, temperature in centigrade, etc. Some questions about the weather, such as "is it raining?" or "is it cloudy?" can be answered using the logical values true and false, while others may need an interpretation of the aforementioned variables. The question "Is it warm?" could relate to the temperature: "If the temperature is above 15°C, it is warm". This would map the temperature in to true/false.

As literals can take the values true and false, so can combinations of literals. Four of the basic combinations, known as *logical connectives*, are *or* ($\vee$), *and* ($\wedge$), *not* ($\neg$), and *implies* ($\Rightarrow$).

When literals are joined using the *and*-connective, they form a *conjunction*, where each part is known as a *conjunct*. Similarly, when joined using *or*, they

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

A	$\neg A$
0	1
1	0

Figure 4.1: Truth tables for four basic connectives. 0 is used to represent *false* and 1 represents *true*.

form a *disjunction*, also know as a *clause*, and the parts are known as *disjuncts*.

The values of combinations of literals, using e.g. the logical connectives, can be specified using a *truth table*. A truth table shows the values, true or false, of all combinations of its arguments. Figure 4.1 show the truth tables for the four mentioned connectives, each connecting two literals, A and B. In these tables the number 1 represents the value true, while 0 represents false.

Logic operations can be defined by their truth tables, and if two operations have identical tables, one may substitute the other. This is known as *reversible substitution*, and it is expressed using a double arrow ($\leftrightarrow$). As can be seen from the truth tables in Figure 4.1 the values for $A \Rightarrow B$ are the same as the values for $\neg A \vee B$. This defines a rule of reversible substitution:

$$A \Rightarrow B \leftrightarrow \neg A \vee B \tag{4.1}$$

The $\neg A$ used in (4.1) is called a *negative literal*, because of the negation, and B is then a positive literal. Clauses with at most one positive literal are known as *Horn clauses*[1], and they form the cornerstone of the use of logic in Prolog. Depending on the number of positive literals, two types of Horn clauses exist: clauses with exactly one positive literal, known as *definite clauses* (see (4.2)), and clauses with no positive literals, known as *goals* (see (4.3)).

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B \quad \leftrightarrow \quad A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B \tag{4.2}$$

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \quad \leftrightarrow \quad A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow \tag{4.3}$$

The interpretation of the definite clause in (4.2) is that B is true only if all the literals $A_1, \ldots, A_n$, on the left-hand side of the "$\Rightarrow$" are true. For the goal in (4.3) a literal should be introduced on the right-hand side of the "$\Rightarrow$", before the expression can be evaluated. If this literal is instantiated, i.e. it has the value true or false, the expression itself can be evaluated to either true or false.

[1]The logician Alfred Horn identified the significance of this type of clauses in 1951.

If the literal is an un-instantiated variable, it should retrieve the value (true or false), which will make the evaluation of the expression return true.

Consider the special case of (4.3) with $n = 2$: $A_1 \wedge A_2 \Rightarrow$. Having $A_1 = true$ and $A_2 = false$, and introducing the literal B on the right-hand side, the expression becomes: $true \wedge false \Rightarrow B$. If B itself evaluates to *true*, the goal is *false* ($true \wedge false \not\Rightarrow true$), while the goal is *true* if the value of B is *false*. If B is a variable it should therefore receive the value *false*, since this would make the goal evaluate to *true*.

4.2.2 Horn Clause Logic

Horn clauses are often written with the positive literal to the left, the direction of the implication arrow reversed, and using commas for *or*, instead of $\wedge$:

$$B \Leftarrow A_1, A_2, \ldots, A_n \tag{4.4}$$

The literals on the right-hand side of the '$\Leftarrow$' constitute the premises (*antecedents*), and the literal on the left-hand side is the conclusion (*consequent*).

In Prolog the names *clauses*, *predicates*, *rules*, and *functions* are all synonymous with Horn clauses. Here the arrow is substituted by a colon and a dash:

```
b :- a1, a2, ..., an
```
(4.5)

The clause in (4.5) can be read as "b is true, if all the values on the right-hand side are true". In Prolog literals starting with a capital letter are interpreted as variables (described later), and the literals used in (4.5) are thus written using minuscule letters.

If a literal serve as the consequent in multiple clauses in a Prolog program, the antecedents may be concatenated using a semi-colon:

```
b :- a1, a2; c1, c2
```
(4.6)

The clause in (4.6) can be read "b is true if a_1 and a_2 are both true, or if c_1 and c_2 are both true". The semi-colon can be interpreted as an *or* connective, and it has a higher precedence than *and*. Parentheses can be introduced to alter the bindings of the colons and semi-colons.

The clauses may be divided into *facts*, *relations*, and *directives*. Common to all of these is that they must be terminated by a full stop. For the readability of this text, the full stop is often omitted.

The building blocks of Prolog are *atoms* and *variables*. An atom is a concrete value, which can be a name (`manpads`), a string (`'Turn left'`) or a numerical value (`600`). All names have a minuscule as the first character.

Variables are used in enquiring about the contents of the facts given (see Section 4.2.2.1). As opposed to atoms variables are always written with a capital as the first character. Variables starting with the underscore ('_') is an exception of this. These are known as *anonymous variables*, since their names can not be referenced. See Section 4.3.2 for an example of questioning with Prolog using atoms and variables.

4.2.2.1 Facts

A fact is a clause with no antecedent, and it is unconditionally true. Facts may be unary, and if so, they are statements about their single argument, such as `ac_type(fighter)` or `altitude(600)`. The interpretation of a fact is determined by the programmer, and these two facts may state that the DSS is used in a fighter aircraft flying at an altitude of 600 m. Here `ac_type` and `altitude` are the names of the facts, while `fighter` and `600` are arguments to them. When more facts share the same name, each of these are said to be an *instance* of the fact.

If a fact has two or more arguments, it describes a relation between these arguments. An example of such a fact is `guidance(manpads, ir)`, declaring that MANPADS are using IR guidance. Prolog has no requirements to the order of arguments in a relation, and it is up to the programmer to keep the order of the arguments in similar facts. If the first argument in the `guidance` fact is defined to be a threat, and the second is a guidance system, then the `guidance(manpads, ir)` is a valid fact, with respect to this definition. Since *Command* is a guidance system and *SA-2* is a threat the `guidance(command, sa2)` fact is not valid. Prolog has no understanding of guidance and it will not know valid facts from invalid facts. If it is asked about e.g. the guidance systems with these two facts, it would give `ir` and `sa2` as answers.

4.2.2.2 Predicates

A predicate has the antecedents-consequent structure as seen in (4.5) and (4.6). Whereas a fact is unconditionally true, a consequent in a predicate is only true if the antecedents are true. The antecedents can be either facts or predicates, or they can be comparisons of results from arithmetic operations.

The example below illustrates the use of both facts and predicates in describing the weather:

```
% Facts about the weather
precipitation(rain).
weather(cloudy).
temperature(21).

% Predicates, describing the weather
it_is_warm :-
    temperature(T),
    T > 15.

weather_is_good :-
    it_is_warm,
    weather(sunny),
    not(precipitation(_)).
```

The predicate `it_is_warm` will evaluate to true, since a fact is stating that the temperature is above 15°C. Since it is raining and not sunny, the predicate `weather_is_good` will return false. The `%` mark a comment, and anything placed to the right of this is not interpreted by the Prolog interpreter. `not` is a standard Prolog-procedure, which is described later.

4.2.2.3 Lists and Structures

In Prolog a list is a data structure containing a number of elements. The list is described within square brackets; the empty list is denoted `[]`, and non-empty lists have a head element and a tail, where the tail itself is a, possibly empty, list. The head has one or more elements, and it is separated from the tail by using the vertical bar: (`[Head|Tail]`). To reference e.g. the third element in a list, which is also the third element in the head of the list, one writes `[_,_,Third|_]`, and the element is referenced by the variable named `Third`. Elements in a list may be atoms, variables, lists, or structures.

A *structure* is a collection of attributes used to describe objects. If a structure is used to describe a person, the attributes may be the person's name, age, and gender. Describing Tom, who is a 33 year old male, can then be done by `person(tom, 33, male)`. A family consisting of a mother, a father, and a number of children can be described as a structure of persons `family(person(_,_,female), person(_,_,male), [_|_])`. Here the list at the end (`[_|_]`) describes the children, and since this description has at least one head element, the family has one or more children.

Structures can be used in goals, just like atoms or variables. In the example above, the goal of finding families, where the father's name is Tom, can be formulated as `family(person(tom,_,_),_,_)`, and finding families with exactly two children is done by: `family(_,_,[_,_])`. As can be seen from these examples, working with structures often deal with the structures of data, rather than the contents of these structures.

4.2.2.4 Directives

Directives are used to make the Prolog interpreter perform various standard operations, such as input/output, generating lists, etc. The number of directives available varies from one Prolog implementation to another. Some of the standard directives, used in the Prolog program described in 4.5, are explained here

The `:-include(<filename>)` directive is used to include the contents of other Prolog files into an embedding file. When this is met by the interpreter, the content of the file, given as argument, is read and interpreted, as was it part of the embedding file.

`findall`, `bagof`, and `setof` are procedures that will collect instances, fulfilling certain criteria, into a single list. `findall` and `bagof` are equivalent, and will both collect all instances into the list, thus allowing for multiple instances of elements in the list. The `setof` procedure will produce a set of the instances, where each element of the set is only present once.

To output text to the screen the `write` procedure is used. If the argument to the procedure is an atom (e.g. a string) it is written as it is, and if it is a variable, the value of this is written. To put a line break in the output the `nl` directive is used.

The `not` predicate will return the negated value of its argument. If the argument is a goal that can be fulfilled, using the `not` procedure will return `no`.

4.3 Answering Questions with Prolog

Prolog is used to perform two different, but related, tasks: describing a domain, and enquiring about it. In its simplest form the first task is done by writing the Prolog program in one or more files, which can then be read by a Prolog interpreter, while the second task may be managed using a Prolog interpreter prompt.

This section describes how one may retrieve information using Prolog, and what a Prolog interpreter does to provide the information.

4.3.1 Matching

A question to the Prolog interpreter, also known as a *goal*, takes on the form of a predicate, or a combination of predicates, and it may contain both atoms and variables. The Prolog interpreter will try to make a *match* between the goal and the predicates given in the Prolog program.

A goal and a predicate match if they are either identical of if variables within them can be instantiated so they become identical. If this can be made the interpreter either returns `yes` or the necessary values of any variables used in the goal that will result in a match. If no match can be found, the interpreter returns `no`.

Consider a Prolog program consisting of these three facts only:

```
precipitation(rain).
weather(cloudy).
temperature(21).
```

If the goal `precipation(rain)` is given to the interpreter, it will return `yes`, since a match can be made between the goal and the first fact in the program. Asking `precipation(P)` will have the interpreter return `P = rain`, since instantiating the variable `P` with the value `rain` will make the match. The goal `wind(breeze)` will not match any of the facts in the program and the answer is `no`. The same answer is returned if the goal is set to `weather(sunny)`, although a predicate named `weather` is part of the program.

Multiple goals can be given at the prompt. The goals are separated by a comma, if all goals should be fulfilled, or a semi-colon if matching one of the goals is

sufficient.

The Prolog program above, describing the weather, can e.g. be used to determine whether one wants to ride the bike to work. Suppose the precipitation can be described using the atoms `snow`, `rain`, `sleet`, and `fog`. Now the goal `precipitation(snow); precipitation(rain); precipitation(sleet)` would get the answer `yes` if either of the sub-goals can be matched. If only the combination of sleet and snow would make the bike stay at home, the goal `precipitation(snow), precipitation(sleet)` should be used.

4.3.2 Working with Prolog

Several Prolog interpreters exist for working with Prolog on a standard PC. The one used in this work is B-Prolog, which offers a prompt interface for asking question about the Prolog program. For more information about B-Prolog see Appendix E.

When working with B-Prolog the clauses are given in a number of files, which are consulted by the Prolog interpreter, before it can provide information about their contents. Suppose a file describes the *guidance* relation using the following clauses:

```
guidance(sa2, command).
guidance(sa3, command).
guidance(stinger, ir).
guidance(stinger, uv).
```

At the Prolog prompt the goal `guidance(Threat, Guidance)` is given. In natural language this should be interpreted as the question: "which threats are using which guidance system?" The interpreter then produces the answer:

```
Threat = sa2
Guidance = command ?
```

The question contains two variables, `Threat` and `Guidance`, and the answer given is the first match found. The question mark at the end of the answer is the Prolog prompt. To get further matches, a semi-colon can be entered at this prompt, and the interpreter then provides the next match:

```
Threat = sa3
```

```
Guidance = command ?
```

When no more matches can be made the interpreter replies `no` when a semi-colon is entered.

Asking the question `guidance(stinger, Guidance)`, with the single variable `Guidance`, produces the following answers (notice the semi-colon after the first two answers):

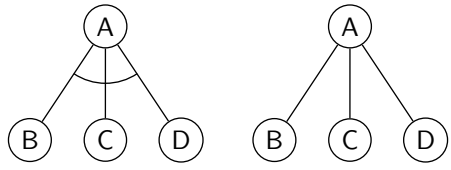
```
Guidance = ir ?;
Guidance = uv ?;
no
```

When asking a question without using variables, or using only anonymous variables, Prolog will simply answer `yes` or `no`, depending on whether or not a match can be found.

4.3.3 Search Trees

To understand how the Prolog interpreter infer the answers to give, it may be helpful to use a graphical representation of the Prolog program, or parts hereof. The graphical representation described here shows the *search tree*, and it reflects the interpreter's internal representation of the Prolog program.

A search tree has two types of nodes: AND nodes and OR nodes. The nodes are drawn with edges to their children, who are also AND/OR nodes. An arc is drawn across the edges connecting the AND node with its children. Parent nodes are drawn above children nodes, and the edges are not directed. Figure 4.2 shows both an AND and an OR node.



(a) AND node (b) OR node

Figure 4.2: AND and OR nodes in a search tree.

Using the AND and OR nodes a Prolog program can be drawn as a tree. While this tree does not show special constructions, such as directives or structures, it gives the possibility to interpret relations at a glance.

The Prolog predicate below is used in the Prolog-based DSS to determine the lethal distance to a threat, based on the kind of countermeasure that should be used in mitigating the threat.

```
lethal_dist(Threat, Dist) :-
    use_cm(Threat, chaff),
    Dist > 500,
    Dist < 5000
    ;
    use_cm(Threat, flares),
    ir_mode(preemptive)
    ;
    use_cm(Threat, flares),
    ir_mode(reactive),
    Dist > 100,
    Dist < 1000.
```

The search tree of this predicate is shown in Figure 4.3. Suppose an enquiry using the goal `lethal_dist(sa5, 1250)` is made. A match is found at the root node, if either of its children nodes matches. The SA-5 threat is using RF guidance, which can be mitigated using chaff. This means that the left branch is the only one that needs to be investigated. The left child node is an AND node which will only return a match if `Dist > 500` (its left node) and `Dist < 5000` (its right node). Since the distance is 1250 (second argument in the goal), both of these will be matched, and the interpreter will return `yes`.

When given a goal the Prolog interpreter will traverse the search tree. Prolog distinguishes between AND nodes and OR nodes. For an AND node to return a match, all of its children must match, while for OR nodes it is sufficient if one of its children provides a match. A node with a single child node may be interpreted as either an AND node or an OR node; it will return a match if the single child node match.

4.3.3.1 Tracing

In e.g. debugging a Prolog program one would benefit from knowing what the Prolog interpreter do to find its answers. To help with this the Prolog command

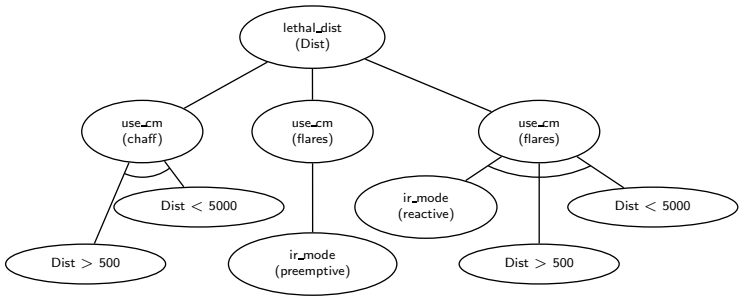


Figure 4.3: A graphical representation of the `lethal_dist` predicate.

`trace` is convenient. After this command is given, the interpreter will write all of its calls to the screen, indicating if they exit with a match or fails.

Extend the weather example from Section 4.2.2 with the following rule and facts:

```
% Clothes to wear
is_clean(t_shirt).
is_clean(shorts).
is_clean(trousers).

what_to_wear(C) :-
    weather_is_good,
    is_clean(C),
    C \= trousers
    ;
    is_clean(C).
```

Using the `trace` command, followed by the `what_to_wear(C)` goal, results in the following trace output:

```
| ?- what_to_wear(C).
 Call: (0) what_to_wear(_72c) ?
   Call: (1) weather_is_good ?
     Call: (2) it_is_warm ?
       Call: (3) temperature(_85c) ?
       Exit: (3) temperature(21) ?
       Call: (4) 21>15 ?
       Exit: (4) 21>15 ?
```

```
     Exit: (2) it_is_warm ?
     Call: (5) weather(sunny) ?
     Exit: (5) weather(sunny) ?
     Call: (6) precipitation(_82c) ?
     Fail: (6) precipitation(_82c) ?
   Exit: (1) weather_is_good ?
   Call: (7) is_clean(_72c) ?
   Exit: (7) is_clean(t_shirt) ?
   Call: (8) t_shirt\=trousers ?
   Exit: (8) t_shirt\=trousers ?
 Exit: (0) what_to_wear(t_shirt) ?
C = t_shirt ? yes
```

The result above states, that if the weather is good, i.e. the temperature is above 15°C, the sun is shining, and there is no precipitation, one should wear a t-shirt, provided it is clean.

The command `notrace` stops the tracing of the Prolog interpreter.

4.3.3.2 Cuts

When the Prolog interpreter traverses a search tree, it may backtrack and search for another match, when a match is found or when no match is found at the bottom of the search tree. The cut operator, `!`, is introduced to prevent Prolog from backtracking, when a match is found. The interpretation of the cut is, that all branches to the right of the `!` are removed from the search tree, when it is met.

Let the `is_clean` fact, as introduced previously, be changed to:

```
is_clean(t_shirt).
is_clean(shorts) :- !.
is_clean(trousers).
```

The search tree for this fact can be seen in in Figure 4.4. In trying to find a match to the goal `is_clean(C)` the interpreter will first seek a match in the left branch, successfully returning `C = t_shirt`. If asked to look for more matches, the second branch is tried, returning `C = shorts`. Since the second instance of the fact contains a cut, all branches to the right of the second branch, i.e. the branch with the `trousers` atom, are cut from the search tree. Asking

for another match will make the interpreter return `no`. Setting the goal to `is_clean(trousers)` will return `yes`, since no matches are made traversing the first two branches, and the cut is thus not invoked.

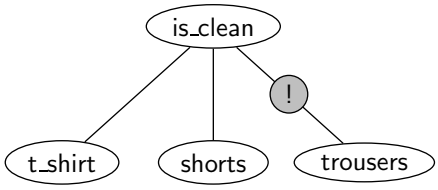


Figure 4.4: The search tree for the `is_clean` fact, after the cut has been executed.

If the cut operator is placed in every predicate that might return a match, it will ensure that at most one match is found. This match will always be the first found, and since predicates are written in a given order in a Prolog file this order will influence the results of the Prolog program. This conflicts with the declarative nature of Prolog, and the fact that understanding the effects of a cut often requires knowledge about the order in which predicates are interpreted, makes the cut an operator that should be used with care.

4.4 Using Prolog for Decision Support

In writing a Prolog program for decision support in fighter aircraft the first step is to define a number of rules for the program to obey. These rules are defined in natural language, and they describe the physical nature of different threats, their guidance systems, and appropriate countermeasures and evasive actions to mitigate these threats. Rules for determining the type of a threat, if any, and spatial relations with friendly aircraft are defined as well. The set of rules in natural language used in developing the Prolog program described in Section 4.5 can be found in Appendix B.1.

The second step is writing Prolog predicates related to the natural language rules. This is done using *stepwise refinement*, where each Prolog predicate may reference predicates not yet defined, or where previously defined predicates may be re-named, re-modelled, or even deleted. Finally all predicates are reviewed, and lacking predicates are defined. This is done in several steps to ensure that all relevant predicates are defined, and that irrelevant predicates are removed from the program. The result is a Prolog program implementing the rules written

in the first step, or a relevant subset hereof, that will suggest actions to any scenario described.

The Prolog program contains current warnings from the MWS and the RWR. Whenever a new warning occurs from either of these, the DSS should be consulted to see which action it proposes. Neither changes in the number or positions of warnings from the RWR nor new warnings from the MWS, do necessarily imply the presence of a new threat as stated in Section 3.5.2.

4.4.1 Assumptions

The program describes countermeasures and threats, which in substance behave as described in Chapter 2. To ease the implementation of the Prolog program, some assumptions are made to the behaviour of threats and countermeasures. Also assumptions about descriptions of angles and distances are made.

The aircraft may be equipped with any combination of the following countermeasures: flares, DIRCM, chaff, jammer, and towed decoy. While the system encompasses all of these countermeasures, they need to be installed on the aircraft, before the DSS will suggest actions involving them. The amount of remaining flares, chaff, and decoys are relevant to the Prolog program. These numbers are assumed updated in-flight by systems other than the DSS.

A warning from the RWR includes the detected threat type and the direction toward the threat. Even though most RWRs can give an estimated distance to the threat, this distance is not used by the program. For MWS warnings the type of threat is not detected. Instead these warnings give the direction and distance to a threat. Since a passive MWS (see Section 2.4) can not detect the distance to a threat an active MWS is assumed.

All radar warnings represent threats, and friendly radar systems, information of which could be given pre-mission, or supplied by an IFF, are not considered an option.

If warnings from both the RWR and the MWS indicate that a threat occurs in a given direction, this threat is likely to be RF guided, since this is the only kind of guidance detected by the RWR. If a MWS warning occurs at a given direction and no RWR warnings occur in that direction, then an IR guided missile is approaching from that angle. An exception form this occurs when a friendly aircraft is positioned in the direction given by the MWS warning, since this could give false MWS warnings. No other warnings are considered to be false. When a RWR warning occurs, without a coinciding MWS warning, the RWR describes

a tracking jammer, and not a RF guided missile.

Both breaklock zones (see Section 2.5.7) and directions to threats can be described in many ways. Even restricting the description to discrete values offers different options, such as integer valued angles, maritime terminology (e.g. fore, beam, and aft), and aircraft parts (e.g. nose, wing, and tail). It is assumed that describing angles using numbers between one and twelve is sufficient. The numbers relates to the positions on a clock, such that twelve o'clock describes the direction straight ahead, six o'clock is in the opposite direction, and so on.

As with angles, different scales are used to describe distances. Altitude is often described using feet, while metres, kilometres, miles, or nautical miles are frequently used to describe longer, and mainly horizontal, distances. It is assumed that the use of metres for all kind of distances does not affect the use of the program.

4.4.2 Available Information

The information available to the DSS can be divided into four different categories, based on the life span of the information. Here the categories are listed with decreasing life spans:

Background Knowledge. Knowledge about missile types and guidance systems are considered background knowledge. This type of information does not change very often, and it is considered static during a large number of missions.

Mission Specific Knowledge. Before each mission, knowledge about the battle scene, and positions and types of threats can be loaded into the system. This information may originate from intelligence sources.

Situation Description. Information about the locations of friendly aircraft may continuously be received via a data link. This type of information is necessary to recognize false missile warnings caused by friendly aircraft.

Warnings from on-board sensors. When a warning occurs from either the RWR or the MWS, information associated with this warning is fed to the DSS. RWR warnings give a direction to the threat, and information about what kind of threat it is estimated to be. MWS will also give a direction to the threat, combined with an estimated distance.

4.5 The Prolog Program

The Prolog program developed implements most of the rules described in Appendix B.1. The program is divided into two parts, where one part is concerned with responding to actual warnings, and the other part responds to assumptions about the environment, in which the aircraft is flying. A warning response consist of three parts: finding the set of appropriate countermeasures, selecting the relevant program for each of these countermeasures, and calculating the manoeuvre that will bring the threats to the breaklock zone for the countermeasure selected.

For each warning the program may suggest more actions. All of these actions comply with the rules set for the program, and they are not prioritized in any way. A prioritization might be performed before the actions found are presented to the pilot. The actions can be ordered according to an estimate of their survivabilities, and if more actions offer the same survivability, the one requiring the least effort from pilot and aircraft would have the highest priority.

Another use of the Prolog program is to query about e.g. the hostility of the environment, or the countermeasures to use for certain threats, etc.

4.5.1 Files

The Prolog program developed is described in seven files. Some of these files are assumed static during flight, while the contents of others files are dynamically updated. The files are described below and their contents can be found in Appendix B. All files have the `.pro` extension. While this is not a necessary extension of Prolog-files, it makes it easier to recognize the files as such.

`dss.pro` This file includes the main parts of the Prolog-based DSS. It includes rules for estimating and addressing the hostility of the environment, and finding relevant countermeasures for the warnings given. The suggestions of countermeasures depend on e.g. the altitude of the aircraft, the distance to threats, and the availability of the countermeasures. It is assumed that no new operational rules are given during flight, and the content of this file is thus considered static.

`warnings.pro` The current warnings are described here. Warnings from the MWS are described with an angle and a direction to the alleged incoming missile, while RWR warnings are described by angle and type of threat.

Since warnings may occur and disappear at any time during flight the content of this file is highly dynamic.

`current.pro` The current description of the aircraft itself is provided in this file. This includes information about altitude, the amount of remaining expendables, and the modes of available countermeasures. The position of friendly aircraft is also given in this file. As with warnings, the information given in this file is highly dynamic.

`mission.pro` Details about the mission, the countermeasures available to the pilot, and a description of the estimated threat scenario are given in this file. This information is supposed to be given pre-mission, and the file can thus be considered static during the mission. If new information about e.g. the positions and numbers of threats become available during flight it should be possible to update this file.

`cm.pro` For chaff and flares a number of different programs are available. Which program to choose depends on the threat, the number of chaff or flares remaining, the altitude, etc. These programs, as well as breaklock zones for different countermeasures, are described in this file, as are rules for determining whether the countermeasures are currently in effect. Neither countermeasures nor their related programs are subjects to frequent changes, and this file can be considered static.

`threats.pro` This file describes background knowledge about all threats that may be encountered, not just the threats expected in the current mission. It also describes what type of guidance systems missiles associated with the threats use, and how to mitigate these guidance systems. This file is considered static.

`util.pro` This file contains functions for handling lists, writing messages to the screen, and calculating manoeuvres between angles. It should be updated only if the DSS itself changes.

The file `dss.pro` includes all other files and it is thus the only file one needs to consult to run the program. The facts and predicates described in the program are listed in Tables 4.1 and 4.2. When the current situation is described in the files `mission.pro`, `current.pro`, and `warnings.pro`, the rule `go` (without arguments) can be invoked at the Prolog prompt. Most of the work is done in the `what_to_do` function, and `go` only measures how long it takes to perform `what_to_do`, and writes the result to the screen.

The program will return all feasible actions. This has the effect that warnings with more than one appropriate countermeasure will get actions recommending each of these, and countermeasures with more than one breaklock zone will be recommended with a manoeuvre to each of these zones.

4.5.2 Atoms and Predicates

The atoms used in the program can be described in sets. These sets are listed in Table 4.1. Some of the sets are used in describing the current situation for the aircraft, while the rest are used by the program in determining the nature and hostility of the environment.

Since Prolog does not have a type check, as e.g. C or C++ has, it is possible to use atoms other than the ones described in Table 4.1. Doing this may cause the program to give unexpected results, and care should therefore be taken when e.g. describing scenarios.

Set:	**Atoms:**
A/C type	`fighter`, `transport`
Angles	`one_o_clock`, `two_o_clock`, ..., `twelve_o_clock`
Band	`ir`, `rf`, `uv`
Countermeasures	`chaff`, `flares`, `dircm`, `jammer`, `towed_decoy`
Decoy Mode	`deployed`, `not_deployed`
DIRCM Mode	`auto`, `receive`, `off`
Environment	`friendly`, `hostile`
Fly mode	`take_off`, `cruise`, `landing`
Guidance	`command`, `sarh`, `qas_tvm`, `inertial`, `ir`, `aclos`, `saclos`, `optical`, `laser_beam_rider`, `uv`
IR mode	`preemptive`, `reactive`
IR threat	`none`, `moderate`, `severe`
Jammer mode	`auto`, `receive`, `off`
Programs	`flare01`, `flare02`, `flaredef`, `chaff01`, `chaff02`, `chaffdef`, `default`
RF Hostility	`low`, `high`
Threats	`sa2`, `sa3`, ..., `roland`, `stinger`, `manpads`
Warnings	`mws`, `rwr`

Table 4.1: The atoms used by the Prolog program. Atoms only serving as strings for output, as well as numbers, are not included in the table.

The predicates defined in the program are listed in Table 4.2. Each of the predicates can be included in a goal, to examine the states leading to the answer given by the DSS. The full program listings can be found in Appendix B.

Predicate:	**Description:**
`ac_type(AC)`	The type of aircraft (`fighter` or `transport`).

Table 4.2: Predicates defined in the Prolog program. *Continues...*

Predicate:	Description:
altitude(Alt)	Altitude in metres.
approp_list(Angle, Cms)	Cms is the list of countermeasures to counter the threats at angle Angle.
available(Cm)	Is the countermeasure Cm available?
breaklock(Cm, Angle)	The Cm has breaklock zone at Angle.
chaff_disp(S)	Chaff was dispensed S seconds ago.
chaff_left(N)	N is the number of chaff cartridges remaining.
cm_has_effect(Cm)	Is Cm currently having effect?
count(N, List)	The List contains N elements.
count_ir_threats(N)	N is the number of probable IR threats in the scenario.
decoy_mode(Mode)	The towed decoy can be either deployed or not_deployed.
dircm_mode(Mode)	The DIRCM can be in one of these modes: auto, receive, or off.
doppler(SA)	The SA threat is a Doppler radar.
flares_disp(S)	Flares was dispensed S seconds ago.
flares_left(N)	N is the number of flares remaining.
fly_mode(Mode)	The mode of flight (take_off, cruise, or landing).
friend(Angle)	A friendly aircraft is positioned at Angle.
go	The main predicate. The time it takes to find solutions is measured here.
guidance(T, G)	The threat T uses a G guidance system.
handle_warning(W)	Suggest an action to counter the warning W.
ir_mode(Mode)	Response mode for IR guided threats (pre-emptive and reactive).
ir_threat(Status)	The Status of the IR threats may be none, moderate, or severe, depending on the number of probable IR threats.
jammer_mode(Mode)	The jammer can be in one of these modes: auto, receive, or off.
lethal_dist(Cm, D)	The countermeasure Cm should only be applied, if the distance (D) to the threat is within the lethal distance of the Cm.
manoeuvre(F, T, D, S)	A manoeuvre from the angle F to the angle T requires S steps in the direction D.
manoeuvre_left(F, T, S)	Manoeuvring from the angle F to the angle T turning left requires S steps.
manoeuvre_right(F, T, S)	Manoeuvring from the angle F to the angle T turning right requires S steps.

Table 4.2: Predicates defined in the Prolog program. *Continues...*

Predicate:	Description:
memberof(M,List)	M is a member of List.
mitigates(B, Cm)	Threats using guidance in the B band can be mitigated using the countermeasure Cm.
phys_guidance(G, B)	The guidance system G works within the band B.
prog(Cm, Prog)	The countermeasure Cm can be activated using the program Prog.
proper_cm(Angle, Cm)	Cm is one of the countermeasures to be used against threats at Angle.
recommend_action(W, Cm, M, P)	The recommended action to the warning W consists of a countermeasure Cm, a manoeuvre M, and a program P.
recommend_cm(W, Cm)	Cm is a recommended countermeasure counting threats at the angle described in the warning W.
recommend_man(W, Cm, M)	The manoeuvre M, described by a direction and a number of steps, should accompany the countermeasure Cm to counter threats at the angle described in the warning W.
rf_hostility(H)	Depending on the amount of probable RF threats in the scenario, the RF hostility can be either low or high.
safe_altitude(B)	The aircraft may fly at an altitude, where threats operating in the band B do not pose a threat.
threat_probable(SA)	The presence of threat SA is considered probable.
turn_left(X, Y)	Angle Y is one step to the left of angle X.
turn_right(X, Y)	Angle Y is one step to the right of angle X.
warning(S, (Angle, Data))	S indicates the sensor from which the warning comes, and Angle gives the direction to the threat. If S is mws the warning describes a missile, and the Data part gives the distance to the missile. If S is rwr the Data part describes the type of threat.
what_to_do	This predicate will find actions related to all warnings, and to the hostility of the environment.
write_cm(Cm, Prog)	The recommended countermeasure Cm and the program Prog to use is written to the screen.

Table 4.2: Predicates defined in the Prolog program. *Continues...*

Predicate:	Description:
`write_manoeuvre(Man)`	Write a description of the manoeuvre `Man` to the screen. A manoeuvre consists of a direction and a number of 30° steps.
`write_threat(Warning)`	Write information about the `Warning` to the screen.

Table 4.2: Predicates defined in the Prolog program.

4.5.3 Comments

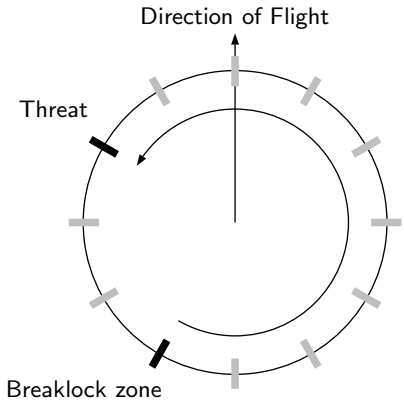
While the program is intended to be self-explanatory, the understanding of some of the predicates may require a few comments.

Even though the purpose of the program is to describe feasible actions for a fighter aircraft when addressing hostile environment or threats, it takes only minor adjustments to make the system work for transport aircraft as well. Therefore the fact `ac_type(<`*aircraft type*`>)`, stating what type of aircraft the program suggest actions for, is introduced. The aircraft type can be either `fighter` or `transport`. This fact is used in the `ir_mode(pre-emptive)` predicate, since a transport aircraft is vulnerable during take-off and landing in enemy territory. Fighter aircraft usually do not take-off and land in enemy territory.

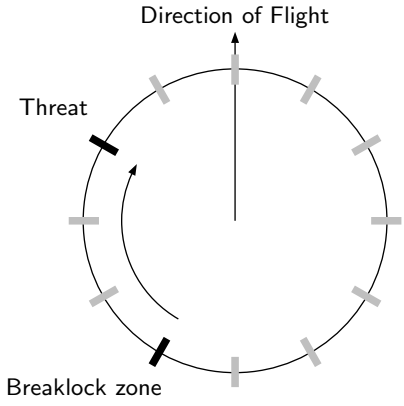
The `safe_altitude(Cm)` predicate will return `yes` if the aircraft is flying in a safe altitude with regards to the countermeasure `Cm`. This predicate only contains safe altitudes regarding missiles using guidance working within the IR and RF bands. For missiles using guidance within the UV band, this predicate will return `no`. While this type of missiles may have a safe altitude, the system works cautiously since it does not reject threats about which it has no knowledge.

`cm_has_effect(Cm)` returns true if the countermeasure `Cm` is currently having an effect on threats. Jammer, towed decoy, and DIRCM should all be turned on to have an effect, while both chaff and flares should have been dispensed within the last few seconds to maintain their effect.

Manoeuvres are used to turn the aircraft around, thus placing threats in the relevant breaklock zones. Figure 4.5 shows the angles and direction involved in doing this. A manoeuvre is described by its direction and a number of steps in that direction. A step is 30°, which is the angle between two consecutive numbers, e.g. one o'clock and two o'clock.



(a) Turning left



(b) Turning right

Figure 4.5: To get the threat within the breaklock zone the aircraft has to perform a manoeuvre. The angle to turn is the angle between the breaklock zone and the threat. Turning both left (Figure 4.5(a)) and right (Figure 4.5(b)) will position the threat within the breaklock zone. Only the turn with the smallest angle is given by the Prolog program.

In [12] it is recommended that long functions should be avoided in Prolog programs, since they are generally difficult to understand. With a few exceptions this principle is followed in the program. One of these exceptions is the `proper_cm` predicate, where each of the instances describe the use of a single countermeasure. To increase the readability of these clauses parentheses and indentation are used.

4.6 Testing

Testing the program must ensure two things. The first is that invoking the `go` rule with any given scenario, the program will suggest all the actions fulfilling the set of rules given in Appendix B.1. The second is that there must be only necessary actions suggested, i.e. all actions must be related to a threat in the scenario or to the hostility of the environment.

Using only the atoms described in Table 4.1, at most one threat of each type at each angle, and a limited set of numeric values to describe e.g. the altitude and distances to threats, there exists a finite number of scenarios to test. This is, however, not a feasible approach, since the number of scenarios may become very large, and the differences between some scenarios are insignificant.

Another approach is to test the predicates individually, and finally test the combination of the predicates in the Prolog program. Testing a predicate like `ir_threat` can be done by using it as a goal with a variable as argument. This variable gets instantiated to one of three atoms, `none`, `moderate`, and `severe`, depending on the number of probable IR threats declared, and the test is done by declaring a number of threats that instantiate the variable to each of these atoms. The results of this test can be seen in Table 4.3.

Number of IR threats:	**Expected status:**	**Instantiation as expected:**
0	`none`	✓
2	`moderate`	✓
5	`severe`	✓

Table 4.3: Results for testing the `ir_threat` predicate.

Predicates working on numbers, such as `count_ir_threats`, can not easily be tested with all possible numbers. To test `count_ir_threats` a number of probable threats are declared, some IR threats and some not. If the predicate is used as a goal, with a variable as argument, and this variable gets instantiated

according to the number of IR threats, the predicate works as intended. For predicates working on lists, the same problem occurs: not all lists can easily be tested for, and a subset should be chosen. The predicate `approp_list` is one example of such a predicate. If used as a goal, with the second argument being a variable, the instantiation of this variable can be controlled to see whether the countermeasures suggested are according to the set of rules.

The testing of all predicates is performed during the development of the Prolog program. To test if the program itself behaves as expected, a number of scenarios are described, and the program is run with each of these scenarios as described below.

4.6.1 Scenarios

Eight different scenarios are used in testing the DSS. These scenarios vary in gravity, from a single threat to a scenario where the number of threats probably exceeds that of any real-world scenario. The last of these scenarios is constructed to be a worst case scenario, and it is included here to find the maximum running time of the DSS. While worse scenarios may be constructed it is assumed that no real-world scenario will require more performance of the Prolog program than this. The scenarios are described in Table 4.4. If nothing else is mentioned in the description of a scenario the fighter aircraft will be equipped with all the described countermeasures, the amounts of available expendables will be high enough to perform any of the programs, and the aircraft is cruising at an altitude of 600 metres. The threats shown in the scenarios are also the threats found to be probable, described using the `threat_probable` predicate.

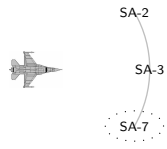
Description:	Scenario:
Scenario 1 A single non-Doppler radar is positioned at twelve o'clock. No missile is detected.	
Scenario 2 This scenario contains three threats, two radar-based (SA-2 and SA-3), and a single threat with IR guidance (SA-7). The threats are all placed in front of the aircraft (eleven o'clock, twelve o'clock, and one o'clock). A missile is fired from the SA-7.	

Table 4.4: Scenarios used for testing the Prolog based DSS. *Continues...*

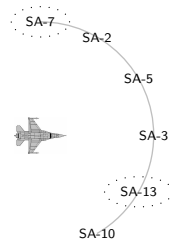
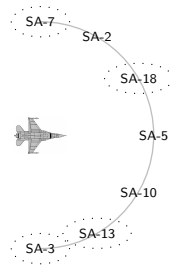
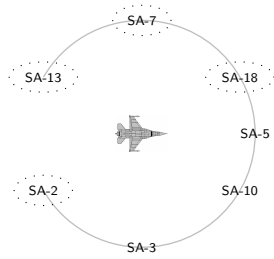
Description:	Scenario:
Scenario 3 Six threats are positioned in front of the aircraft, covering all positions from nine o'clock to two o'clock. IR guided missiles are being launched from positions at nine o'clock (SA-7) and one o'clock (SA-13).	
Scenario 4 Seven threats positioned in a semicircle in front of the aircraft. All IR guided threats are launching missiles (SA-7, SA-13, and SA-18). Also a single RF guided missile is launched from a SA-3 at three o'clock. The jammer and the towed decoy are unavailable on-board the aircraft.	
Scenario 5 IR guided missiles are launched from positions at seven o'clock, nine o'clock, and eleven o'clock. RF based threats are positioned at twelve o'clock, one o'clock, three o'clock, and five o'clock. The threat at five o'clock, a SA-2, is also launching a missile. The aircraft altitude is 250 m.	
Scenario 6 The threat scenario is the same as in scenario 5. Here the aircraft altitude is 1000 m.	
Scenario 7 The threat scenario is the same as in scenario 5. Here the aircraft altitude is 2500 m.	

Table 4.4: Scenarios used for testing the Prolog based DSS. *Continues...*

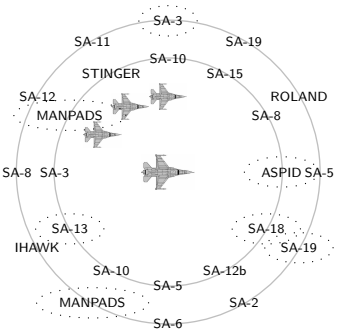
Description:	Scenario:
Scenario 8 – Worst case This scenario contains a large variety of threats. The threats are positioned at all angles and at different distances. Both IR and RF guided missiles are launched. Three friendly aircraft are positioned at seven o'clock, eight o'clock, and nine o'clock)	

Table 4.4: Scenarios used for testing the Prolog based DSS. For each scenario the placements of threats are depicted. Threats launching missiles against the aircraft are marked with a dotted oval.

For each of the scenarios described in Table 4.4 a file named `scenarioX.pro` is made. This file includes both the static files (`dss.pro`, `cm.pro`, `threats.pro`, `util.pro`), and the dynamic files (`warnings.pro`, `current.pro`, `mission.pro`), which are renamed to reflect the scenario (`warningsX.pro`, `currentX.pro`, `missionX.pro`). In all file names the `X` is replaced by the number of the scenario. A description of the results running the Prolog program for each of these scenarios can be found in Table 4.5

Testing the Prolog program with the described scenarios revealed minor flaws with the program. One of these concerned the use of breaklock zones. When a threat was to be countered by a given countermeasure the program will suggest a turn to bring the threat within the breaklock zone of that countermeasure. Although the jammer breaklock zone was described to span from eleven o'clock to one o'clock no suggestions were given to bring the threat to the twelve o'clock angle. It turned out that the clause `breaklock(jammer, tvelwe_o_clock)` had a misspelled atom (`tvelwe_o_clock` instead of `twelve_o_clock`). To the Prolog interpreter the misspelled atom was considered to be a new atom since atoms do not need to be declared before being used. Correcting the misspelled atom made the program suggest turns to the twelve o'clock angle when the jammer was recommended.

Another flaw was found when a MWS warning was given, and no friendly aircraft was registered. When finding actions to mitigate a MWS warning the program

Scenario:	Warnings:	Running time:	Threat responses:	Scenario responses:	As expected:
1	1	7.1	Use chaff, jammer, or towed decoy.	Use jammer in auto mode.	✓
2	3	18.0	Use flares, chaff, jammer, or towed decoy.	Use jammer in auto mode.	✓
3	6	24.2	Use flares, chaff, jammer, or towed decoy. Do not use chaff to counter SA-5 and SA-10.	Use jammer in auto mode.	✓
4	8	10.0	Use chaff to counter RF guided missile and RF threats. Use flares to counter IR guided missiles.	Use chaff (default program).	✓
5	8	5.0	Use flares to counter all missiles. Do not use countermeasures against RF threats.	No responses.	✓
6	8	28.0	Flares against IR guided missiles. Chaff, jammer, and towed decoy. No chaff against SA-5 and SA-10 (Doppler).	Use jammer in auto mode.	✓
7	8	26.0	No flares to counter IR missiles. Chaff, jammer, towed decoy to counter RF threats. No chaff against SA-5 and SA-10 (Doppler).	Use jammer in auto mode.	✓
8	23	113.0	Use appropriate countermeasures. Not able to distinguish between threats at same angle.	Use flares (default program) and jammer in auto mode,	✓

Table 4.5: The results of testing the Prolog program with eight different scenarios. Running time is an average over ten runs. It is given in milliseconds.

first checks if a friendly aircraft is reported to be at the same angle and if so the warning is neglected. If no friendly aircraft was registered the B-Prolog interpreter was not able to run. This may be an interpreter dependent error since other interpreters may just return `no` if a predicate has not been defined. The flaw was corrected by always defining a `friend(none)` clause, and the program can now be run by the B-Prolog interpreter.

As no more flaws were found the program returns expected responses to all of the scenarios. Appropriate countermeasures are suggested depending on the threats, the aircraft altitude, and the countermeasures available. Also responses to the scenario are as expected. They depend on the types and number of probable threats in the scenario, and on the availability of countermeasures.

It is found that the number of warnings in the scenario description has some influence on the time it takes to run the Prolog program. The first scenario has one warning only, and finding solutions to this is done in an average of 7 milliseconds. Responses to the last scenario, having 23 warnings, are found at an average of 113 milliseconds. Since four of the eight scenarios have eight warnings, and the average time for finding solutions for these vary from 5 to 28 milliseconds, it can not be concluded that the number of warnings alone determine the time it takes to find solutions to a given scenario.

4.7 Discussion

In developing the Prolog program, the first step was to describe a set of rules for the program to fulfil. During development, questions to these rules come up, and more rules may become necessary. Therefore the set of rules are not to be considered a static entity. The set will evolve in a number of iterations during the development of the program.

Some of the rules are expressed in such a way that they can be more or less directly translated into Prolog predicates. Other rules have a more subtle formulation, which makes them more difficult to implement. The rule "The jammer will reveal the position of the aircraft" is an example of one such rule. While it is true, and relatively easy to implement, the implications of the rule on the Prolog program are not obvious. A rule stating "The jammer should only be active in a non-severe environment if it mitigates a missile launched towards the aircraft", would describe the same intention: the jammer should not be active unless it makes a positive difference to the survivability. Implementing it in the program is just as easy.

Running a Prolog program, compared to running a program written in e.g. C or C++, will often require substantially longer execution time. One reason for this is that when the execution of a program finds an answer to a goal, this answer is not stored, and the next time the same goal is set, the answer will be obtained once again. Another reason is that the Prolog program is often interpreted, which itself is almost synonymous with a prolonged execution time. There exist Prolog methods for self-modification, so that answers to goals that may be used multiple times, are stored as facts. For the program developed here, running on the laptop PC described in Appendix E, the execution time never exceeded 150 ms. To this amount of time the time it takes to process data before and after the Prolog DSS is invoked needs to be added. Processing data constitutes collecting relevant sensor output data from systems on-board the aircraft and preparing them for processing by the DSS, and after a list of relevant actions is found by the DSS, the list must be prioritized and presented to the pilot. Although the total amount of time is not known, it is assumed that for most scenarios it will be less than the 200 ms limit set in Section 3.3. Therefore no measures are taken to improve execution time in this work.

In using this program, an approaching missile can be "hidden" by a friendly aircraft. Even if the MWS recognizes the missile the MWS warning will be considered false if it is placed in an angle similar to that of a friendly aircraft. While this will bring down the number of false alarms, it can not be described as failsafe. If a missile is in fact attacking from this angle, only the aircraft closest to the missile will respond to it. This might be enough to mitigate the approaching missile; but combining the effort of more aircraft could increase the effect on the missile, thus providing all friendly aircraft with a higher survivability. If warnings from the MWS were to be shared among the friendly aircrafts, using e.g. a data link system, this could add to the usability of the DSS. A MWS warning repeated by a friendly aircraft in the same direction may not be ignored and proper evasive actions have to be found.

The order of the declarations of predicates, as well as the order of predicates within other predicates, will have no influence on the results given by the interpreter when a program is interpreted according to the definitions of Prolog. B-Prolog, as well as most other Prolog interpreters, does not comply with this in full. An example of this can be seen in Figures 4.6 and 4.7.

```
recommend_cm(_,(Angle,_),Cm) :-
    approp_list(Angle, Cms),
    memberof(Cm, Cms).
```

Figure 4.6: Implementation of `recommend_cm` that works.

```
recommend_cm(_,(Angle,_),Cm) :-
    memberof(Cm, Cms),
    approp_list(Angle, Cms).
```

Figure 4.7: Implementation of `recommend_cm` that does not work.

When a Prolog program is interpreted, the predicates are tested in the order they appear. In the working version of the `recommend_cm` predicate, shown in Figures 4.6, the `approp_list` is interpreted first, thus instantiating the list of appropriate countermeasures, `Cms`, which is then used in the second predicate. In the non-working version, shown in 4.7, the order of these two predicates is reversed. Hence the interpreter will first try to make a list containing the variable `Cm`. Since `Cm` is not instantiated, there exists no defined list that will fulfil this clause, and the query will not succeed. Whether the Prolog interpreter implementation discovers this, and exits gracefully, or keeps on running until it is out of memory, is entirely up to the implementation. The B-Prolog interpreter does not detect this inexpediency, an continues to look for the countermeasure until it runs out of memory.

Some facts are meant to occur no more than once in the Prolog program. For instance declaring multiple instances of the `altitude` or the `fly_mode` makes no sense. Despite of this, it can easily be done, resulting in a program that does not behave according to the intentions. Declaring more `altitude`s could result in the position of the aircraft being interpreted as out of range of both IR and RF guided missiles. To solve this kind of conflicts the directive `once` may be used. Replacing `altitude(Alt)` with `once(altitude(Alt))` will result in only the first instance of `altitude` being used.

In Section 4.1 it is stated that Prolog programs are easy to read. While this is true, the readability can be enhanced introducing e.g. *operator* descriptions. Mathematical operators, such as + (add), - (subtract), / (divide), · (multiply), or % (percent) are easily recognized. They can be either prefix, (+ and -), infix (+, -, /, or ·), or postfix (%), and they are described by a precedence. The precedence is used to establish the order in which operators are evaluated, staring with the lowest precedence, and since division and multiplication have lower precedence than addition and subtraction, the expression $A + B \,/\, C - D \cdot E$ is equivalent to $A + (B/C) - (D \cdot E)$.

The operator `is_guided_by` can be declared by giving the directive `:-op(600, xfx, is_guided_by)`. It is given the precedence 600 and the `xfx` part describes it as an infix operator. Operators may be prefix (`fx`), infix (`xfx`), or postfix (`xf`). With the `is_guided_by` operator the fact `guidance(sa2, command)` may be written as `sa2 is_guided_by command`, which may be closer to a natural

language description of the guidance relation.

4.8 Conclusion

Developing a DSS prototype using Prolog is relatively easy. Based on rules stated by experts in the EW domain simple Prolog predicates can be developed with little effort. Combining all the rules translated from formulations in natural language into one working Prolog program requires a little more effort. The concept of having the program perform two tasks, one finding proper responses to the environment and one handling warnings from RWR and MWS, is not evident from the original set of rules.

While the concept of imperative programming seems intuitive, it may be difficult to adjust to for a programmer used to imperative programming. Some knowledge about imperative programming may prove useful when programming Prolog, e.g. to find out why a Prolog program runs out of memory, when the order of clauses is not set right.

Having a fixed set of rules to build the DSS from may not be the best foundation to build upon. As the development of the Prolog program progress the set of rules must be updated to reflect new requirements to the knowledge gained from the rules.

In a DSS the use of Prolog will narrow the set of possible actions. While this may help the pilot deciding on the actions to actually perform, it is not guaranteed that action chosen will give the best survivability possible. Through his training the pilot will have learned the rules from which the DSS infer actions, and therefore an experienced pilot may not benefit as much from the DSS as a rookie would. Since the hostile environment changes from one theatre to another, and the tactics to follow are constantly evolving, even the best skilled pilot can benefit from the presence of a Prolog based DSS, provided it is being frequently updated.

Usually Prolog programs are considered slow. The Prolog program described in this chapter performs relatively fast, and solutions are found within the stated 200 millisecond limit. Adding to the usability of the program is likely to slow down the performance of it, and a satisfactory trade-off between usability and performance must be found.

Chapter 5

The Bayesian Network Approach

This chapter describes the use of a Bayesian Network (BN) as a method for evaluating actions for a fighter pilot to perform when a threat occurs. It describes the basic theory of BN and why the technique may be an appropriate approach in designing a DSS for fighter pilots. The process of constructing a BN using the HUGIN tool is described. HUGIN offers a graphical user interface for the construction of the network [7, 28] and details on HUGIN can be found in Appendix E.

The model developed gives the probability of the survival of the aircraft depending on the state of the world surrounding the aircraft, knowledge about an emerging threat, and a selection of possible actions for the pilot to take. In building the model, part of the work may seem both cumbersome and complex. Hence two methods to do these parts in a semi-automatic way are explored and described as well.

BNs are used in a wide range of areas, including vision, natural language processing, robot navigation, planning, and machine learning [17]. In relevant literature BNs are also referred to as *belief networks*, *Bayesian belief networks*, *Bayesian dependency nets*, or *causal probabilistic networks*. The name *decision graphs*, as used in section 5.2.6, refers to BNs containing *utility* and/or *decision* nodes

[22]. These are also known as *influence diagrams* [17]. See [13, 21, 22, 23, 34] for more in-depth description of BN and probabilistic reasoning.

5.1 Motivation

A decision support system on-board an aircraft will depend on information from a number of sources. The uncertainty of some information (such as the kind and amount of expendables on-board the aircraft, the pattern in which they will be dispensed, and the angle in which they will depart from the aircraft) may be considered negligible. Information about the kind of missiles the aircraft is likely to encounter may be obtained from intelligence sources, and should be considered with some degree of uncertainty. Finally, data from on-board warning systems, giving angle, distance, and speed between the aircraft and an incoming missile, must be considered with high degrees of uncertainty. Incorrect or incomplete data may be all the DSS initially has to support its decisions.

Since the sensors on-board the aircraft do not give an accurate image of the world surrounding it, using a BN to model this world seems a plausible approach as a BN can deal with the probabilities of e.g. the sensors being wrong and the countermeasures not working as intended.

5.2 Basic Theory

A BN consists of a set of random variables, each variable having a number of mutually exclusive states (at least two), and the variable can be in any one of its states with a given probability. It can be depicted as a directed acyclic graph, $G = (V, E)$, with V being the set of variables shown as nodes in the graph, and the set of directed edges between nodes, E, indicating dependencies between them. Changes in states of some variables may cause changes in states of other variables. The strengths of dependencies between variables are represented in tables named *dependency tables*, *conditional probability tables*, or *potential tables* [22, 34]. In this work the term *dependency table* is used. For nodes without ancestors, the dependency tables contain unconditional probabilities.

Relations between nodes in a BN are depicted using arrows. If A and B are variables in the same network modelling a given domain an edge from B to A indicates that changes in the probabilities of the states in B may result in changes in the probabilities of states in A. A is then said to be a child of B, and B is a parent of A (see Figure 5.1). A variable can have a number of both

children and parents. The set of parent nodes to A is written as $pa(\mathsf{A})$, and the family set containing A and its parent nodes is written as $fa(\mathsf{A})$.

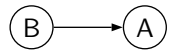


Figure 5.1: States in A depends on states in B.

In a BN arrows can be both causal and non-causal. When arrows are causal changes in the states of the real-world entity represented by the parent node may cause changes in the states of a child node representing another real-world entity. When non-causality is used the causality may be directed against the arrow for some relations while it follows the arrow for other relations in the same network. While this is perfectly legal when constructing a BN it may be difficult to maintain and develop a model if no clear causal relations are given by the arrows. For modelling purposes the edges should therefore always indicate the causality between nodes.

Dependencies between states in A, $a_1, \ldots, a_n$, and states in B, $b_1, \ldots, b_m$, can be described in a dependency table as shown in Table 5.1. In this table each row describes the dependencies between a state in A (a_1 or a_2) and the states in B (b_1 and b_2). The states of a variable are mutually exclusive and the probabilities for each column must sum up to 1, i.e. the probability for a variable being in none of its states is zero.

B	b_1	b_2
a_1	0.2	0.87
a_2	0.8	0.13

Table 5.1: Dependency table for a node A showing its dependencies of states in its parent node B.

When constructing the BN all nodes will have *prior* probabilities. For a node without parent nodes these probabilities can be based on e.g. observations while nodes with parents have prior probabilities dictated by their dependency tables and the probability distributions of their parents. When a variable receives *evidence* a new probability distribution based on e.g. recent observations is given to it, independent of prior distributions. If the probability of the variable being in a given state is 1 after the state has been given evidence, the variable is said to be *instantiated* in that state. When one or more variables receive evidence the probabilities for all depending states in other variables in the network are updated.

The nomenclature used in this chapter has been collected in Table 5.2.

A, B	Nodes in the BN
a_i, b_j	States of a node
$pa(\mathsf{A})$	The set of parents to nodes in the set A
$fa(\mathsf{A})$	The family set including A and its parents
$P(a)$	The probability of the state a
$\mathsf{A} \perp \mathsf{B} \mid \mathsf{C}$	A and B are d-separated given evidence to C

Table 5.2: Nomenclature for Bayesian networks.

5.2.1 Probability Calculations

The probability of A being in state a_i is written as $P(\mathsf{A} = a_i)$. When the variable involved is clearly defined by the context the form $P(a_i)$ is used. The probability of A being in state a_i depending on B being in state b_j is written as $P(\mathsf{A} = a_i|\mathsf{B} = b_j)$ (or $P(a_i|b_j)$ for short).

For doing probability calculations the *fundamental rule* is given by:

$$P(a_i|b_j)P(b_j) = P(a_i, b_j). \tag{5.1}$$

Here a_i, b_j means that A is in state a_i and B is in state b_j. Since $P(a_i|b_j)P(b_j) = P(a_i, b_j) = P(b_j|a_i)P(a_i)$ the following rule is formulated:

$$P(b_j|a_i) = \frac{P(a_i|b_j)P(b_j)}{P(a_i)}. \tag{5.2}$$

This rule is known as *Baye's Rule*[1] and it is fundamental to the use of Bayesian networks. The probability $P(a_i|b_j)$ clearly indicates that the value of A depends on the value of B. If the value of A is known, this probability also indicates some knowledge about the value of B. $P(a_i|b_j)$ is thus often referred to as the *likelihood of* b_j *given* a_i and it is written as $L(b_j|a_i)$.

The HUGIN software uses *potentials* instead of probabilities. A potential can be any non-negative real number. The potential distribution for a given variable can be turned into a probability distribution by *normalization*. Let A be a variable with n states, and let $\pi(a_i)$ be the potential of the ith state of A. The probability $P(a_i)$ of A being in the state a_i is then given by

$$P(a_i) = \frac{\pi(a_i)}{\sum_{j=1}^{n} \pi(a_j)}.$$

[1] The rule is named after the Presbyterian minister and mathematician Thomas Bayes, 1702 – 1761, who came up with the formulation "A is known given knowledge about B"[3].

5.2.2 d-separation

When one or more nodes in a BN have received evidence, the prior probability distribution in the BN is no longer valid, and a new probability distribution is to be propagated throughout the net. In propagating evidence the *d-separation* property between pairs of nodes in the net is vital. This section gives the definition of d-separation and in Section 5.2.5 a use of d-separation in propagating the probability distributions within a BN is described.

When changes in the states of a node A have no influence on the node B in the BN the nodes are said to be d-separated (short for "dependency separated"), and this is written $A \perp B$. The connection between two nodes, and the nodes connecting them, determines when these nodes are d-separated. If the two nodes are d-separated given evidence to some node C this is written as $A \perp B|C$. When two nodes are not d-separated they are said to be *d-connected*.

All connections between two nodes can be classified as either *serial*, *converging*, or *diverging*. For a serial connection the two nodes are d-separated if any node between them has become instantiated. In Figure 5.2 the nodes Influenza and Fatigue are d-separated if Fever has become instantiated. At first glance it might seem counterintuitive that Fatigue and Influenza are not related given Fever; if a person has a fever he or she can still be tired due to the flu. The definition of d-separation does not describe the state in which the intermediate node is instantiated. Thus if the person does not have a fever, any fatigue is no longer related to the presence of a flu.

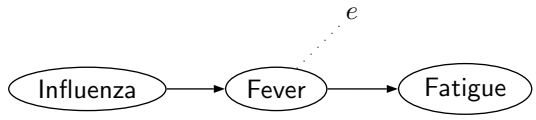


Figure 5.2: d-separation in a serial connection.

In Figure 5.3 a diverging connection between nodes in a BN is shown. Here the two nodes, Fever and Headache, are d-separated if their common parent Influenza has been instantiated. To understand this, assume that the person does not have the flu. Now the person can have both a fever and a headache, but they are not related.

In diverging connections nodes are d-separated when intermediate nodes, or their descendants, have not received evidence. In Figure 5.4 a fever can be caused by both Influenza and Cold. If evidence has shown that a person has a

Figure 5.3: d-separation in a diverging connection.

fever, Influenza and Cold becomes d-connected.

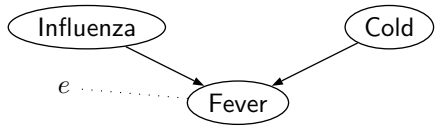


Figure 5.4: d-separation in a converging connection.

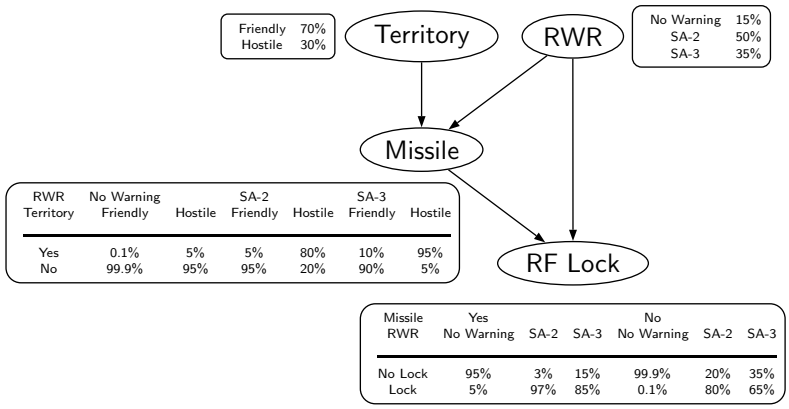
If two nodes A and B are d-separated and evidence e is given to a node in the BN, the probability $P(\mathsf{A}|\mathsf{B}, e)$ is given by $P(\mathsf{A}|e)$.

5.2.3 Joint Probability Distribution

For a BN the Joint Probability Distribution (JPD) is the probabilities of each of the combinations of states in the nodes of the BN. If for instance the BN has two nodes, A with states a_1 and a_2 and B with states b_1, b_2, and b_3, then the JPD is comprised of the six probabilities $P(a_1, b_1)$, $P(a_1, b_2)$, $P(a_1, b_3)$, $P(a_2, b_1)$, $P(a_2, b_2)$, and $P(a_2, b_3)$.

Figure 5.5 shows a small BN describing the dependencies between four nodes. For each node in the BN the dependency table is shown. In the following this BN is used for illustrating the principles of probability calculations.

The probability of the variables being in a given combination can be found using the fundamental rule (5.1). An example of this probability calculation is given by:



Territory		Friendly		Hostile	
RF Lock		No	Yes	No	Yes
Missile = No	RWR = No	0.10479	0.00010	0.04271	0.00004
	RWR = SA-2	0.06650	0.26600	0.00600	0.02400
	RWR = SA-3	0.07718	0.14333	0.00184	0.00341
Missile = Yes	RWR = No	0.00010	0.00001	0.00214	0.00011
	RWR = SA-2	0.00053	0.01698	0.00360	0.11640
	RWR = SA-3	0.00368	0.02083	0.01496	0.08479

Figure 5.5: A small BN describing the dependencies between a RF lock, the presence of a missile, the type of warning coming from a RWR, and the hostility of the territory over which the aircraft is flying. It is assumed that the presence of radar systems does not depend on the state of the territory. A dependency table is shown for each node. The JPD calculated from the dependency tables is shown at the bottom.

$$
\begin{aligned}
&P(\mathsf{RF\ Lock} = \mathit{No}, \mathsf{Missile} = \mathit{No}, \mathsf{RWR} = \mathit{SA\text{-}2}, \mathsf{Territory} = \mathit{Friendly}) \\
&\quad = \quad P(\mathsf{RF\ Lock} = \mathit{No} \mid \mathsf{Missile} = \mathit{No}, \mathsf{RWR} = \mathit{SA\text{-}2}) \cdot \\
&\qquad\quad P(\mathsf{Missile} = \mathit{No} \mid \mathsf{RWR} = \mathit{SA\text{-}2}, \mathsf{Territory} = \mathit{Friendly}) \cdot \\
&\qquad\quad P(\mathsf{RWR} = \mathit{SA\text{-}2}) \cdot \\
&\qquad\quad P(\mathsf{Territory} = \mathit{Friendly}) \\
&\quad = \quad 20\% \cdot 95\% \cdot 50\% \cdot 70\% \\
&\quad = \quad 6.5\%
\end{aligned}
$$

Finding the probability of any combination of states in the BN is easily done by a simple look-up in the JPD. The combinations of states in some nodes, disregarding the probabilities of states in other nodes, may also be found from calculations on the entries in the JPD (see Section 5.2.4). Unfortunately the number of entries grows exponentially with the number of nodes in the BN and for larger BNs it may prove impossible to maintain the full JPD on a computer. For larger BNs it is therefore useful to keep the representation of the probability distribution as a BN and do the probability calculations by propagating evidence throughout the BN (see Section 5.2.5).

5.2.4 Prior and Posterior Probabilities

As stated in Section 5.2.3 the JPD can be used to find the probability of any combination of states. To do this the entries in the JPD associated with that combination are added up. This is known as the *marginal probability*. The marginal probability for $\mathsf{RF\ Lock} = \mathit{Yes}$ from the BN in Figure 5.5 is given by:

$$
\begin{aligned}
&P(\mathsf{RF\ Lock} = \mathit{Yes}) \\
&\quad = \quad 0.00010 + 0.26600 + 0.14333 + 0.00001 + 0.01698 + 0.02083 + \\
&\qquad\quad 0.00004 + 0.02400 + 0.00341 + 0.00011 + 0.11640 + 0.08479 \\
&\quad = \quad 0.67599
\end{aligned}
$$

This means that at any given time, with no evidence for any of the variables in the BN, the probability of a RF lock on the aircraft is 67.6%. As no evidence is entered this number is the *prior marginal probability*.

If a variable gets instantiated finding the *posterior probability* of another variable is done by narrowing the number of entries in the table and then re-normalize it. If e.g. the RWR issues a warning, and the RWR gets instantiated to $SA-2$, all the entries in the JPD with $\mathsf{RWR} = \mathit{NoWarning}$ or $\mathsf{RWR} = SA{-}3$ will represent impossible states. So finding the posterior probability of $\mathsf{RF\ Lock}$ being *Lock* is done like this:

$$
\begin{aligned}
&P(\mathsf{RF\ Lock} = \mathit{Yes} \mid \mathsf{RWR} = \mathit{SA\text{-}2}) \\
&= \frac{\sum P(\mathsf{RF\ Lock} = \mathit{Yes}, \mathsf{RWR} = \mathit{SA\text{-}2})}{\sum P(\mathsf{RWR} = \mathit{SA\text{-}2})} \\
&= \frac{0.26600 + 0.01698 + 0.02400 + 0.11640}{\begin{array}{c}0.06650 + 0.26600 + 0.00600 + 0.02400 + \\ 0.00053 + 0.01698 + 0.00360 + 0.11640\end{array}} \\
&= 0.84675
\end{aligned}
$$

Comparing the prior and posterior probabilities just described one gets that knowing that a RWR warning is issued the probability of a RF lock increases to 84.7% compared to the 67.6% when no knowledge about the RWR was given.

Due to the symmetry of conditional probabilities, updating the BN can be used to both predict the outcome of different settings, and to examine the settings that yield the best outcome. Table 5.3 shows the prior and posterior probabilities for the BN in Figure 5.5. The posterior probabilities shown are calculated when the RF Lock node is instantiated in the *NoLock* state. This instantiation causes an increase in each of the probabilities $P(\mathsf{Territory} = \mathit{Friendly})$, $P(\mathsf{RWR} = \mathit{NoWarning})$, and $P(\mathsf{Missile} = \mathit{No})$.

State:	Prior:	Posterior:
Territory = Friendly	70.00%	78.01%
Territory = Hostile	30.00%	21.99%
Missile = Yes	26.41%	7.72%
Missile = No	73.59%	92.28%
RWR = No Warning	15.00%	46.21%
RWR = SA-2	50.00%	23.65%
RWR = SA-3	35.00%	30.14%
RF Lock = No Lock	32.40%	**100.00%**
RF Lock = Lock	67.60%	**0.00%**

Table 5.3: The prior and posterior probabilities of when instantiating the RF Lock in the *No Lock* state.

5.2.5 Propagating Evidence

When an event occurs in the world modelled by a BN the probabilities of one or more states in the nodes may change. This happens if a node has received

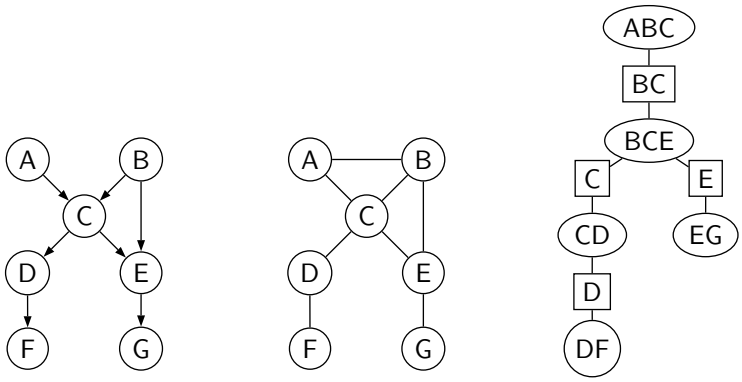
evidence (e.g. the probability of an approaching missile will increases when the MWS issues a warning) or if a node has been instantiated (e.g. when an approaching missile has actually been spotted). When probabilities for states in the BN changes the dependency tables for all relevant nodes needs to be updated throughout the BN. Calculating prior probabilities can always be done using the values of the JPD. For larger networks this is not a feasible solution since the JPD may become too large to be stored in computer memory, and the computations will be too numerous to be carried out in reasonable time. A number of methods have been proposed to both reduce the size of the storage necessary and to speed up the propagation of evidence.

5.2.5.1 Propagation in HUGIN

In HUGIN the propagation of evidence is done using *junction trees*. This section describes how to construct a junction tree from a BN, and how to use it for propagation of evidence. Details about the construction of junction trees and the HUGIN propagation are given in [21].

Before propagating evidence the structure and dependencies of a BN are collected in a junction tree. The junction tree consist of a number of nodes known as *cliques*, each of which represents a number of nodes from the BN. For each node A_i in the BN that has a non-empty parent set, $pa(\mathsf{A}_i) \neq \emptyset$, at least one clique in the junction tree will represent the set $pa(\mathsf{A}_i) \cup \mathsf{A}_i$. Each clique has a table representing the part of the JPD given by the nodes represented by the clique. The construction of a junction tree is done in the following steps:

- From the BN a *moral graph* is constructed. This is done by first adding connections between any parent nodes sharing a common child node, and then removing the directions on all edges. Figure 5.6(b) shows the moral graph made from the BN shown in Figure 5.6(a).
- A *junction graph* is constructed from the moral graph by collecting clusters of nodes from the moral graph, where all nodes in the clusters are interconnected. Each of these clusters constitutes a clique in the junction graph. Cliques that share nodes from the moral graph are connected in the junction graph. Each connection is labelled with a *separator*, shown graphically as a rectangle containing the nodes in common for the cliques it connects.
- Let A and B be two cliques in the junction tree. The path between A and B contains the nodes in common for A and B. This is known as the *junction tree property*. If the junction graph contains no cycles (i.e. it is



(a) The original Bayesian network (b) The moral graph (c) A junction tree

Figure 5.6: From the original BN in (a) the moral graph in (b) is found. Based on this the junction tree in (c) is constructed.

a tree) and the junction tree property is observed, the junction graph is also the junction tree used for propagation.

- If the junction graph is not yet a junction tree, the moral graph gets *triangulated*. In triangulation cycles in the graph are identified. If a cycle has more than three nodes in it *fill-in chords* are added between nodes not connected.

- The junction tree is a subgraph of the junction graph. To dissolve cycles in the junction graph connections are removed as long as the remaining graph obeys the junction tree property. The result of this is the final junction tree. Figure 5.6(c) shows a junction tree for the BN shown in Figure 5.6(a). Note that the ABC clique is arbitrarily chosen as the root node.

The propagation in HUGIN is done using two functions: `DistributeEvidence` and `CollectEvidence`. The algorithms are used on the junction tree made for the BN. When a node in the BN receives evidence the `CollectEvidence` function is called with the root clique of the junction tree as argument. This function will return the evidence entered in the table of the clique given as argument, but not before this table has been updated by calling the `CollectEvidence` function with each of is children cliques as argument. When the root clique of the junction tree has received evidence from all of its children cliques, it updates its table and distributes this to its children. This is done using the `DistributeEvidence`

function, which is used recursively throughout the junction tree. The resulting tables of the cliques now contain the updated probability distribution of the BN.

5.2.6 Decision Graphs

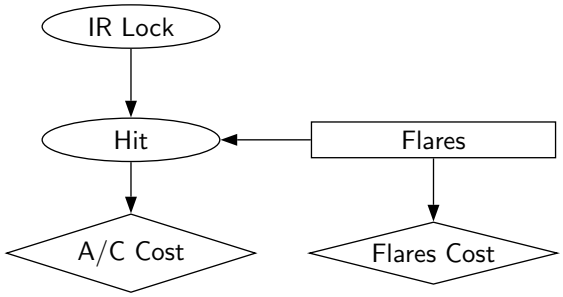
When using a BN for decision support a number of combinations of states having received evidence can be compared. In this comparison each combination can be associated with a scalar value, and the best combination will be the one yielding the highest/lowest value. If the BN is to be used in a DSS for fighter pilots the value to compare solutions by may be the probability of the aircraft surviving an attack (P_S).

In associating a BN with a scalar value two types of nodes are introduced: *decision nodes* and *utility nodes*. Adding a node of either of these types to a BN makes it a Decision Graph (DG) [22], also known as an *influence diagram* [23]. Graphically decision nodes are shown as rectangular and utility nodes are diamond-shaped. A DG showing both decision and utility nodes can be seen in Figure 5.7. In a BN each node represents a variable. For a DG this is not the case since neither action nor utility nodes represent variables in the domain modelled.

Decision nodes represent actions in the network, and they have no dependency tables. Changes in the state of a decision node may influence the probability distribution in ordinary nodes, while there is no influence the other way around.

The utility nodes represents additive contributions to the Expected Utility (EU) of the BN. The EU for a DG is the measure that is sought optimised. It may have a unit and all contributions to it stem from utility nodes. Each utility node is drawn with incoming lines from the nodes that may influence the outcome of that node. Where ordinary nodes have a dependency table a utility node has a table showing the contribution by each combination of states in its parent nodes. Note that this contribution may be negative, e.g. representing a cost in a total turnover.

The DG in 5.7 was constructed using HUGIN. It models the situation of an Unmanned Aerial Vehicle (UAV) (cost price $100,000) flying over enemy territory. The UAV has been equipped with a flare dispenser and the DG can be used to calculate the expected utility depending on whether or not flares are dispensed in a given situation. The EU is the sum of two terms, one concerned with the price of dispensing flares ($1,000 per dispensing), and one concerned with the expenses related to a possible UAV crash. With e being the evidence given to the nodes IR Lock and Hit, and C being the cost of dispensing flares, the EU is



(a) The DG

No Lock	50%
Lock	50%

(b) Dependency table for IR Lock

Flares	Dispensed		No	
IR Lock	No Lock	Lock	No Lock	Lock
Yes	0.0001%	5%	0.2%	98%
No	99.9999%	95%	99.8%	2%

(c) Dependency table for Hit

Hit	Yes	No
Utility	$100,000	$0

(d) Utility table for A/C Cost

Flares	Dispensed	No
Utility	$1000	$10

(e) Utility table for Flares Cost

Figure 5.7: DG modelling the cost of flare dispensing. Dispensing flares influence the total cost of the flight.

given as:

$$EU(\mathsf{Flares}|e) = C(\mathsf{Flares}) + \sum_{\mathsf{Hit}} U(\mathsf{Hit})P(\mathsf{Hit}|\mathsf{Flares}, e)$$

When no knowledge about an IR lock has been received the probability of its presence is set to 50%. The EU of dispensing flares can then be found as:

$$\begin{aligned} EU(\mathit{Dispensed}) &= \$1000 + \$100,000 \cdot 0.0001\% \cdot 50\% + \$100,000 \cdot 5\% \cdot 50\% \\ &= \$3500.05 \end{aligned}$$

This can be compared to the EU of not dispensing flares:

$$\begin{aligned} EU(\mathit{No}) &= \$10 + \$100,000 \cdot 0.20\% \cdot 50\% + \$100,000 + 98\% \cdot 50\% \\ &= \$49,110 \end{aligned}$$

So saving flares while flying over hostile territory (with $P(\mathsf{IR\ Lock} = \mathit{Lock}) = 50\%$) may prove to be very expensive.

If evidence is given stating that no IR lock is present the EU of dispensing is changed to \$1000, while the EU of no dispensing is \$210. So if no knowledge is given about IR locks, and they are present, it is better to dispense flares, just to be on the safe side. On the other hand, if intelligence reports that no possible IR threats are in sight, dispensing flares would just be a waste.

In a general BN the direction of edges does not necessary need to display causality. In a DG they do. Consider the BN shown in Figure 5.2 (repeated in Figure 5.8). Here fatigue is caused by fever which is then again caused by the flu. If the action of taking an aspirin, which is generally taken to lower fever, is given as evidence to the Fever node, it will influence the fever and hence the fatigue. If the directions of the edges were reversed, the aspirin would lower the fever, thus handling the influenza, without helping with the fatigue.[2]

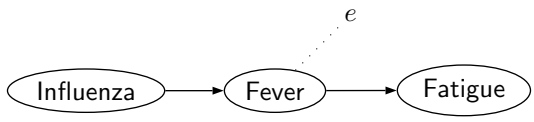


Figure 5.8: d-separation in a serial connection. (Figure 5.2 repeated here.)

[2]Example taken from [22].

5.2.7 Utility Scale

The EU is the sum of utilities from all utility nodes in the BN. Therefore it is necessary for utilities to have a common unit. In maximising profit or minimising cost a monetary unit will be adequate, while the unit to use in other cases may be less obvious. For a company some actions may result in a momentary higher income, while decreasing the customer satisfaction. In the long haul this may lead to a loss of customers, thus decreasing the turnover. In a situation like this the EU should be measured using a *utility scale*, weighing both income and customer satisfaction.

Using a DG for decision support for fighter pilots will also require an appropriate utility scale. Having the probability of survival, P_S, defining the scale may be less opportune since this probability depends on combinations of actions. This means that the values of the utility nodes does not depend solely on states in discrete nodes, and therefore the utility nodes cannot be correctly connected to any nodes in the DG. To remedy this, a single node to connect a utility node to can be constructed. This node will contain all combinations of states from the nodes on which the survivability depends. While this is possible the construction of such a node is cumbersome and the ability for using a DG for modelling the domain is weakened.

If survivability is represented as a score, where mitigating a threat would have a positive influence on the score, while e.g. using expendables or making the threat focus on the aircraft by turning on the jammer when this is not necessary, would have a negative influence, optimising the survivability can be done. Section C describes the construction of a score system that may be used as utility scale.

Instead of using the intangible survivability, one may instead operate on the concept of costs. Everything involved in the mitigation of a threat has a fiscal cost, and finding the combination of manoeuvres and countermeasures that will yield the lowest total cost, can then become the aim of decision making. To find the price of the involved countermeasures is relatively easy since all flares, chaff, towed decoy, etc., has a list price. So does a new aircraft, if the current aircraft is lost to an incoming missile. This price may be harder to find, since it involves a complex mixture of e.g. salary to people employed with the procurement of new aircraft, the inability to fulfil requirements about the number of deployed aircraft, the use of aircraft for training, and so forth. Also finding the price of a new pilot is involved, and although this is possible seen from a pragmatic point of view, it may not be politically correct to do so.

5.3 Building the Model

The model developed gives the probability of the survival of the aircraft, P_S, depending on the states of variables representing the surrounding world, knowledge about an emerging threat, and a selection of possible actions for the pilot to take.

The first step is to specify the structure of the BN based on knowledge about the EW domain. This structure can have several layouts, depending on the degree of details one wishes to model. The general rule is to incorporate enough details to be able to establish the probability distribution between nodes. On the other hand, more details will require more nodes, and thus populating the dependency tables becomes more difficult. When the structure of the BN is in place, every node is supplied with a sufficient number of states, and the conditional probabilities between nodes are entered into the dependency tables.

At first the model is a "pure" BN since no decision or utility nodes are introduced. Any probabilities used in the model are merely based on "good guesses" and not on data from real aircraft and threats. Adding decision and utility nodes to the BN will change it into a DG. As explained in Section 5.2.7 no good utility scale is found for the use of a DG in decision support for fighter pilot, and hence no utility nodes are used in the model.

5.3.1 Assumptions

It is assumed that the current combat scenario contains at most one threat, and that only a single missile may be launched towards the aircraft at any given moment. This will ensure that when multiple nodes receive evidence they are all the result of the presence of a single threat. If more threats are present in the scenario, and they are detected by e.g. radiation in several RF bands, the probability of a proper detection of each of these by the BN decreases.

In the Prolog approach (Chapter 4) it is assumed that all warnings relate to real threats or friendly aircraft. In the BN approach this is not the case. Here it is assumed that all observations come with a probability/certainty, and thus a warning indicates that a threat is present with a given probability. Since none of the on-board sensors/warners give this number in conjunction with a warning it is up to a *threat evaluator* to calculate it. This can be done using statistics on the number of times a warning was given for a threat in a given distance and at a given angle, etc. This threat evaluator is assumed existing and working, and the design of it is not considered part of this work.

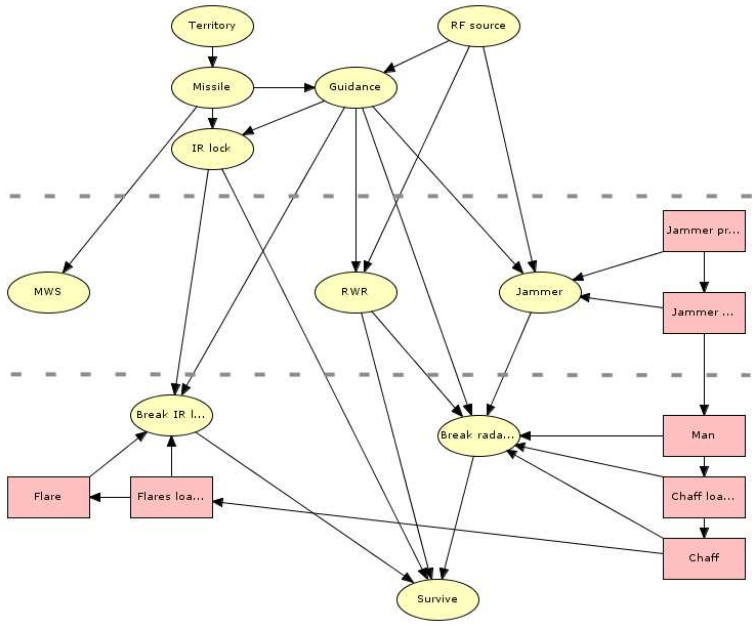


Figure 5.9: The BN modelling the world in and around the fighter aircraft. The model is divided into three layers, one representing the world surrounding the aircraft, another representing the on-board sensors, and the last one giving the survivability for a given combination of states having received evidence. (Picture exported from HUGIN.)

5.3.2 Constructing the Model

The BN constructed can be seen in Figure 5.9. It is lay out as a three layer model, with the top layer representing the world surrounding the aircraft, the middle layer containing nodes for the on-board sensors, and finally the bottom layer giving the results of deploying countermeasures. The jammer is placed in the middle layer, as it gives input about RF locks. Since it serves as both a sensor and a countermeasure it might as well have been placed in the bottom layer.

All edges between the ordinary nodes in the BN shows causality. To ease the reading of the model the nodes are arranged in such a way that causality points downwards. In this way the nodes in the top layer (the surrounding world), causes changes in the states of the nodes in the middle layer (the sensors), which in turn influence the calculated survivability.

In the construction of the model some observations are made. A number of these are described below.

Enemy Territory. If enemies are present and threatening the aircraft, the aircraft is, by definition, flying over enemy territory. This is why the Enemy Territory is not just a binary node, indicating whether or not the border has been crossed; enemies might exist on the friendly side of a border as well (e.g. terrorists), and missiles may be "friendly fire", i.e. launched by ones own forces.

Missile Seen. It is possible for the pilot to visually see an incoming missile that he has not been warned about by either the MWS or the RWR. To represent this, a Missile Seen node was initially added to the BN model. While the presence of a missile that has not been seen by the aircraft sensors is of vital importance to the results of the DSS, it may not be possible to use this information in the DSS. To do so would require the pilot to tell the system that a missile has been seen, and possibly both the direction and range to the missile. While it is possible to construct and incorporate this type of registration in the PVI, the registering process in itself would be too time consuming to be feasible. The missile may have hit the aircraft before the pilot has told the system about it. For this reason the node is not part of the model.

Expendables. In the model dispensing chaff or flares is described using a binary action node; either they are dispensed or they are not. In the real world there would be more programs to select from, each program designed to take care of a given threat or threat scenario. Introducing more programs into the model will increment the number of entries in the dependency tables, which again will make it more difficult to find proper values for making the BN behave properly.

Guidance System. It is assumed that any missile guidance system uses either RF or IR guidance, and that none of the missiles under consideration in this work has multiple guidance systems. Therefore the detection of a radar guided missile, as seen by the RWR can intuitively lead to the assumption that a missile detected by the MWS, assuming it is detected in the same direction, will not be using IR guidance. This gives the causality between two nodes, RF Guidance and IR Guidance, as seen in Figure 5.10.

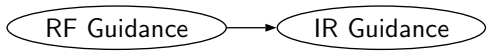


Figure 5.10: If a missile is RF guided it will not also be IR guided.

Since RF and IR guidance are mutual exclusive there is no reason to maintain two nodes, and the RF Guidance and IR Guidance nodes are thus merged to a single node named Guidance.

Since different types of missiles use the same type of guidance, with different results, the presence of an IR lock depends on both the type of missile and on the guidance in use. It can not depend on the missile type alone, since missiles having the same missile types may be equipped with different types of guidance systems.

Decision and Utility Nodes A number of decision nodes are added to the model. These nodes and their states can be seen in Table 5.4. Let $|\mathsf{A}|$ be the number of states in a decision node A. With the seven nodes in the model there are $|\mathsf{Jammer\ Present}| \cdot |\mathsf{Jammer\ Mode}| \cdot |\mathsf{Manoeuvre}| \cdot |\mathsf{Chaff\ Loaded}| \cdot |\mathsf{Chaff}| \cdot |\mathsf{Flares\ Loaded}| \cdot |\mathsf{Flares}| = 2 \cdot 3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 288$ combinations of states in the decision nodes.

Not all decision nodes need to be part of the decision since the pilot has no way of changing the state of these while in-flight. The decision nodes is thus split into two sets, the *preparation nodes* comprising the Jammer Present, Chaff Loaded, and Flares Loaded, and the *action nodes*: Jammer Mode, Manoeuvre, Chaff, and Flares. This now gives a total of $3 \cdot 3 \cdot 2 \cdot 2 = 36$ combinations of states in the action nodes which should be tested to find the combination yielding the highest survivability. Some of these combinations would not be feasible, and could thus be removed from the set of combinations to test for. For instance a combination of flares and manoeuvres does not make any sense, if Flares Loaded indicate that no flares are available.

Decision node:	**States:**
Jammer Present	Yes, No
Jammer Mode	Off, Rx, Auto
Manoeuvre	None, Left, Right
Chaff Loaded	Yes, No
Chaff	No, Dispense
Flares Loaded	Yes, No
Flares	No, Dispense

Table 5.4: The decision nodes and their states.

As can be seen in Figure 5.9 the action nodes are interconnected. In Section 5.2.6 it is stated that action nodes have no dependency tables, and the edges between action nodes does not constitute a parent-child relation per se. Instead they are needed for the propagation algorithm used by HUGIN. The direction of the arrows between the action nodes is of no influence to the calculated survivability.

As described in Section 5.2.6 the introduction of a utility scale for the model is not straightforward. At first the number of remaining expendables was introduced as a utility scale and utility nodes were connected to the Flare and Chaff nodes. This is skipped since flares and chaff are two very different types of expendables, and having a number of flares left will not increase the survivability when a RF based threat occurs. Second, optimizing the amount of inventory would only result in using less expendable, not in increasing the survivability at all. For these reasons no utility nodes are used in the final model.

5.4 Populating Dependency Tables

The work with constructing a BN can be divided into two parts: the construction of the qualitative structure of the net, where all nodes and dependencies between nodes are established, and the quantitative population of the dependency tables. Different approaches to ease the latter part of the process have been tried. In [17] the relations between nodes are described using Prolog-like Horn clauses and by parting the network into self-contained objects, and in [24] the degrees of certainty in relations are described using simple sentences in semi-natural English. While these approaches have some advantages in the initial population of dependency tables, they do not appeal to the strength of using a precision tool as BN. The HUGIN tool offers different techniques for describing the probability distributions using e.g. discrete distributions or arithmetic functions.

5.4.1 Multi-dependency Tables

In this work a simple model describing the domain is developed. This model contains the nodes deemed necessary for decision support for fighter pilots, and each of the nodes has a very limited number of states. In simple tests the simple model works as expected, but it does not handle the complexity of e.g. several different missile types and guidance systems. Therefore the model is expanded by adding more states to some of the nodes. Going from e.g. two values for the Missile node (Present/Not present) to 12 values (Not present and 11 different missile types) gives not only six times as many entries in the Missile node dependency table, but also in the tables of all nodes dependent on the Missile node.

This section describes a semi-automatic procedure for producing multi-dependency tables, i.e. tables giving the dependency of multiple parents. For nodes hav-

B	b_1	b_2
a_1	$B(1,1)$	$B(1,2)$
a_2	$B(2,1)$	$B(2,2)$

Table 5.5: The B table.

ing more than one parent, dependency tables are created for each of the parent nodes. The cells of these dependency tables are then multiplied with each other, assuming that states in any two parent variables can be treated as independent. This gives an initial value in the multi-dependency table.

To see how this works let A, B, and C be three nodes, with A being a child of both B and C, as shown in Figure 5.11.

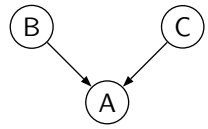


Figure 5.11: A is dependent on B and C.

If A has the states $a_1, \ldots, a_n$, B has the states $b_1, \ldots, b_m$, and C the states $c_1, \ldots, c_l$, then $P(\mathsf{A}|\mathsf{B})$ is an $n \times m$ dependency table (referred to as B) and $P(\mathsf{A}|\mathsf{C})$ is an $n \times l$ dependency table (referred to as C). The values in these tables are combined in an $n \times (m \cdot l)$ table BC where the value at position (i, j), $i = 1, \ldots, n$, $j = 1, \ldots, m \cdot l$, is calculated as $B(i, \lceil j/l \rceil) \cdot C(i, j \bmod l)$. Tables 5.5, 5.6 and 5.7 show the dependency tables for $n = 2$, $m = 2$, and $l = 3$.

C	c_1	c_2	c_3
a_1	$C(1,1)$	$C(1,2)$	$C(1,3)$
a_2	$C(2,1)$	$C(2,2)$	$C(2,3)$

Table 5.6: The C table.

The table BC does not equal $P(\mathsf{A}|\mathsf{B},\mathsf{C})$, although it gives an approximation to it. An example to illustrate, that BC will not always be a valid dependency table follows. The B and C tables shown in Tables 5.8 and 5.9 are both valid dependency tables, with the probability in each column of the two tables summing up to 1. If the cells in these tables are multiplied according to the algorithm previously described the resulting BC table is the one shown in Table 5.10.

As can be easily seen BC is not a valid dependency table since five of the

B	b_1			b_2		
C	c_1	c_2	c_3	c_1	c_2	c_3
a_1	$B(1,1)\cdot C(1,1)$	$B(1,1)\cdot C(1,2)$	$B(1,1)\cdot C(1,3)$	$B(1,2)\cdot C(1,1)$	$B(1,2)\cdot C(1,2)$	$B(1,2)\cdot C(1,3)$
a_2	$B(2,1)\cdot C(2,1)$	$B(2,1)\cdot C(2,2)$	$B(2,1)\cdot C(2,3)$	$B(2,2)\cdot C(2,1)$	$B(2,2)\cdot C(2,2)$	$B(2,2)\cdot C(2,3)$

Table 5.7: The BC table.

B	b_1	b_2
a_1	1	0.25
a_2	0	0.75

Table 5.8: The B table with values.

six columns do not sum up to 1. For the rightmost three columns this can be handled by normalizing the values. Let BC^* be the normalized table with $BC^*_{i,j} = BC_{i,j} / \sum_{k=1}^{n} BC_{k,j}$, where i and k denote rows, and j denotes the column. For the two leftmost columns this is not applicable since each of these columns sum up to 0. By replacing each cell in these columns by a 1 before normalizing, the resulting cells will all have the value of $1/n$. The probability interpretation of this is that "no combination of A, B, and C states is possible" has been replaced by "all combinations of A, B, and C states have the same probability".

5.5 Structural Learning

As seen in Section 5.2.3 a BN is a way of representing a JPD in an intuitive way. Populating the dependency tables requires a full JPD, or knowledge about how to obtain it. If this knowledge is not available, methods exists to learn the structure and dependencies in a BN from sample data.

Structural Learning (SL) is a method to automatically construct a BN on the base of sample data faithfully representing the scenario to be modelled. Variants of this method are described in e.g. [22] and [53]. The process of performing SL with the HUGIN tool is illustrated in Figure 5.12 and described in Section 5.5.1. Section 5.6 describes how a BN is build using SL based on synthetic data from a missile approach simulator.

C	c_1	c_2	c_3
a_1	0	0	1
a_2	1	1	0

Table 5.9: The C table with values.

B	b_1			b_2		
C	c_1	c_2	c_3	c_1	c_2	c_3
a_1	0	0	1	0	0	0.25
a_2	0	0	0	0.75	0.75	0

Table 5.10: Result of multiplying cells from B and C.

5.5.1 Structural Learning using HUGIN

Constructing a BN using the SL feature in HUGIN is done in three major steps: preparing data from the domain, learning the structure of the BN, and finally finding the probability distributions in the dependency tables of the nodes in the BN. Each of these steps are divided into several sub-steps, as can be seen in Figure 5.12. The three steps are described below.

Data Preparation. At first the data source is selected. Data may be read from a text file, formatted as a table, where each column represents a domain parameter, or it may be read from a relational database. If a database is selected all joins between tables need to be described, so a joint table can be generated. In the final part of this phase it is possible to define replacements or discretisation of values for the given parameters, or to exclude variables from the BN to be generated.

Structural Learning. Each of the parameters determined in the data preparation phase will be represented by a node in the constructed BN. The first step in this phase is to describe known dependencies/independencies between these nodes. The actual SL, which is the next step, will use this predefined knowledge in constructing the BN. To perform the SL HUGIN offers the use of two different algorithms, named PC and NPC. The PC algorithm is briefly described in section 5.5.2. Both of these algorithms need a preset level of significance for detected dependencies, and this can be set before performing the SL step. After this step any structural uncertainties found by the algorithm in use can be manually solved. As the last step in this phase, HUGIN lets the user see how strong each of the detected dependencies is.

Probability Distribution. To fill in the dependency tables a method named

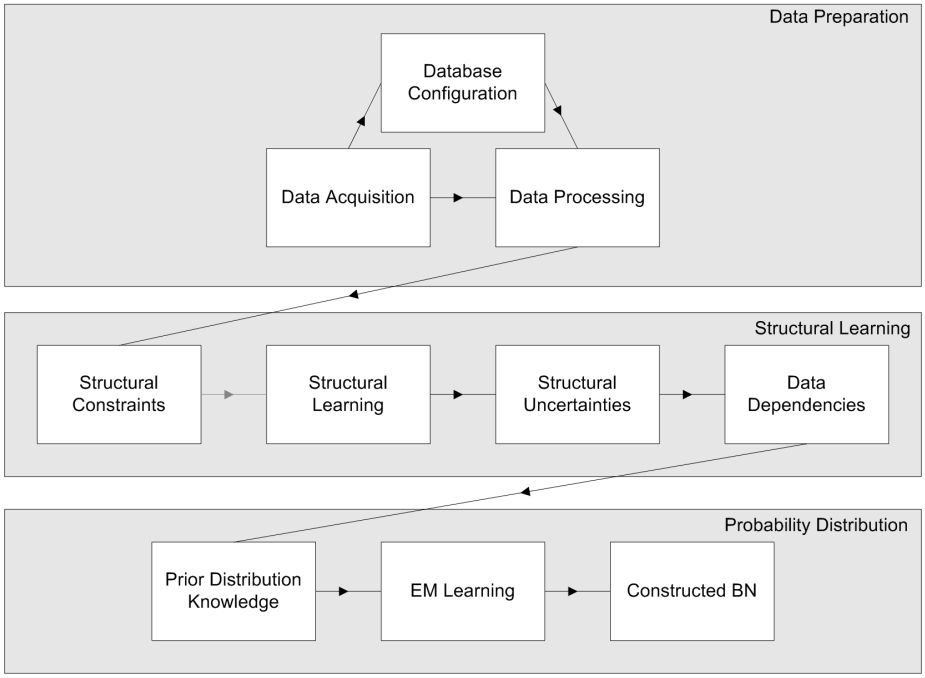


Figure 5.12: Structural Learning using HUGIN. The boxes indicate the steps involved in constructing a BN using the SL feature in HUGIN.

Estimation-Maximization (EM) is used. This method is described in section 5.5.3. It uses the distributions of node values given in the sample data. If other distributions of the parameters are to be used, it is possible to set these distributions prior to the EM-step. The EM-method will make the contents of the dependency tables reflect the distributions of the given parameters in a number of iterations. Both the number of iterations and a convergence threshold can be set using the HUGIN user interface.

5.5.2 The PC Algorithm

A BN consists of a set of nodes V and a directed acyclic graph G that connects the nodes via a set of edges. An effective SL algorithm will find a graph G that describes the relations obtainable from the sample data, without examining all combinations of edges between nodes in V. Assuming that two nodes, A and B, may have one of four different relations (A depends on B, B depends on A, A and B depends on each other, or A and B are independent) the number of possible

relations in a graph with n nodes is $4\binom{n}{2}$. Given that no cycles are allowed in a BN reduces this number of combinations, although the number will still be too large to make an exhaustive search through all of them feasible. For a network with 12 variables the number of directed acyclic graphs is approximately $5.4{\cdot}10^{39}$ [45].

With the HUGIN tool the graph can be constructed using either the PC[3] or the NPC algorithm[4]. The BN described in this work is constructed using the PC algorithm, and hence it is the one described here. Descriptions of the algorithm can be found in [28, 44, 45].

The PC algorithm performs four steps in learning the structure of a BN:

1. Find the conditional dependencies between nodes in the BN. This is actually done by finding all the pairs of nodes that are conditionally independent, and then categorizing the pairs left as conditionally dependent. The dependency between any two nodes, A and B, is examined given a set of nodes S_{AB} not including A and B. S_{AB} of sizes ranging from 0 to 3 are used. If A and B are conditionally independent given S_{AB}, $\mathsf{A}\perp\mathsf{B}|S_{\mathsf{AB}}$, the search for independency between A and B is halted, and an independency relation between them is registered. This is tested by statistical tests using a significance level set using the HUGIN user interface before initiating the algorithm.

2. Identify the skeleton of the graph from the dependencies and independencies found during step 1. The skeleton is the graph consisting of all the dependencies without directions.

3. Find the graph colliders; these are nodes where arrowheads from multiple arrows collide, i.e. they depend on multiple nodes. The colliders are found using one rule: Consider three nodes, A, B, and C, where A and B are connected, B and C are connected, and A and C are not connected. If $\mathsf{B} \notin S_{\mathsf{AC}}$ for any S_{AC} satisfying $\mathsf{A}\perp\mathsf{C}|S_{\mathsf{AC}}$, then B is a collider. This is illustrated in Figure 5.13.

4. Supply directions to all edges, thus making the graph directed. This is done by repeated application of four rules to the edges in the graph. These four rules are illustrated in Figure 5.14. The direction of the resulting arrow in the first rule follows from the fact that no collider was found. The second, third, and fourth rules ensure that no cycles are introduced to

[3]The algorithm was described by Peter Spirtes and Clark Glymour and the letters PC are the initials of the authors' first names. The PC algorithm is developed from the SGS algorithm described by P. Spirtes, C. Glymour, and R. Scheines.

[4]The NPC algorithm is an extension of the PC algorithm. The extension consists of a *Necessary Path Condition*, hence the name.

(a) Before step two (b) After step two

Figure 5.13: Identifying B as collider.

the graph. The fourth rule is only necessary if known dependencies/independencies are introduced to the structure of the BN before the learning process is initiated. The dashed line indicates that the nodes A and C has a registered independency.

5.5.3 The EM Algorithm

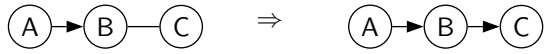
A BN is described by a graph, G, showing the nodes and their interdependencies, and a set of parameters, Θ, describing how states of a given node depend on the states of the parents of this node. The PC algorithm generates a G for the BN, and the EM algorithm is used to estimate Θ, thus populating the dependency tables of the BN. The algorithm consists of two steps, the estimation step (E-step) and the maximization step (M-step). These are used alternately to produce adequate dependency tables. The algorithm is run for a number of iterations, each iteration containing both an E- and an M-step, until some given threshold is reached. With HUGIN this threshold is either the number of iterations or a maximum difference between the results of two contiguous steps. Both thresholds can be set in the HUGIN user interface. The algorithm is described in detail in [25, 26, 28].

Let θ_{ijk} describe the dependency between a state k in the node X_i, and the state j of $pa(\mathsf{X}_i)$:

$$\theta_{ijk} = P(\mathsf{X}_i = k \mid pa(\mathsf{X}_i) = j)$$

The set Θ is the set of dependencies, $\Theta = \bigcup_{ijk} \theta_{ijk}$. The family set $fa(\mathsf{X}_i)$ of a node X_i is given as $fa(\mathsf{X}_i) = \mathsf{X}_i \cup pa(\mathsf{X}_i)$.

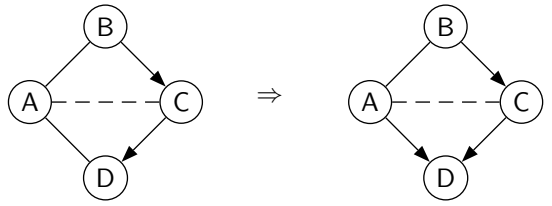
If data used in the parameter learning phase are complete, and faithfully cover all situations possible for the BN to be constructed, the number of different combinations of states within $fa(\mathsf{X}_i)$ may be used to calculate the probability distribution of states in X_i. Since a BN is often constructed based on a more sparse sample data set *expected counts* are used instead. In the E-step the



(a) First rule

(b) Second rule

(c) Third rule



(d) Fourth rule

Figure 5.14: The four rules for adding orientation to edges in the BN during SL.

expected counts n^* for each family $fa(\mathsf{X}_i)$ and parent $pa(\mathsf{X}_i)$ configuration of every node X_i is computed based on the current counts n. This is done using the function E_Θ based on the current set of parameters Θ, and the sample data D. Some details on the E_Θ function can be found in [25].

$$\begin{aligned} n^*_{fa} &= E_\Theta\{n_{fa}(\mathsf{X}_i = k, pa(X_i) = j)|D\} \\ n^*_{pa} &= E_\Theta\{n_{pa}(pa(\mathsf{X}_i) = j)|D\} \end{aligned}$$

In the M-step n^*_{fa} and n^*_{pa} are used to update the parameters θ^*_{ijk}, thus forming a new set of parameters $\Theta^* = \bigcup_{ijk} \theta^*_{ijk}$:

$$\theta^*_{ijk} = \frac{n^*_{fa}}{n^*_{pa}}$$

In the next iteration the "old" values in the algorithm are replaced by the "new" values: $\Theta \leftarrow \Theta^*$, $n_{fa} \leftarrow n^*_{fa}$, and $n_{pa} \leftarrow n^*_{pa}$.

5.6 Generating Data with Fly-In

A software package named Fly-In is used to generate sample data for the SL of a BN to be used in the EW domain. The Fly-In software simulates the flight of an IR guided missile towards an aircraft. It does so by taking models of an aircraft, a missile, the image processing used in guiding the missile, and flares dispensed from the aircraft, and combine these with motion models for aircraft and missile to calculate the resulting missile approach. The calculations are done in contiguous time steps where the resulting positions of missile, aircraft, and possibly dispensed flares in one time step are used as input for the next time step. For each time step the image that might be seen by an IR camera at the tip of the missile is generated. The generated image is then analyzed to find the possible target points for the missile to aim for. When flares are dispensed they are also added to the generated image, and they will hence be included as possible targets in the simulation of the missile approach. Concluding each time step calculation the change in the trajectory of the missile is found, and aircraft, flares, and missile are moved according to their motion models, before calculations for the next time step is initiated. Some information on the Fly-In software is given in Appendix E.

Each missile approach simulation is ended when either a preset maximal amount of flying time is reached, or when the missile hits or passes the aircraft. If the missile passes the aircraft Fly-In gives the estimated smallest distance between

them. Since a missile might be proximity fused the distance is used in the later data processing.

Fly-In is used for simulating 600 missile approaches. Doing this on the laptop PC described in Appendix E takes several days. A simulation is defined by numerous parameters, all influencing the result. The simulations used in this work include combinations of six parameters only, and thus the BN constructed from the results of the simulations can contain seven nodes only, one for each parameter, and one describing the result of the simulation. The resulting BN is depicted in Figure 5.15.

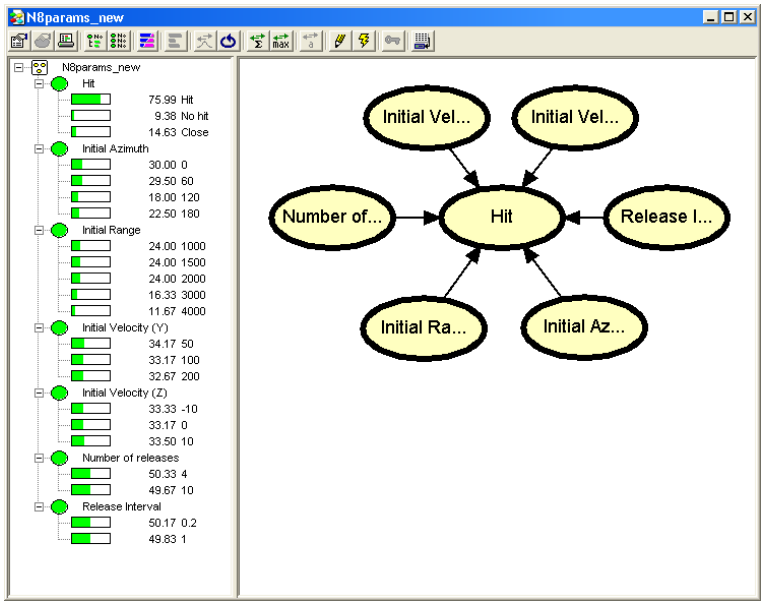


Figure 5.15: The BN generated from Fly-In data. (Screenshot from HUGIN.)

Since the Fly-In software can only do missile simulations for IR guided missiles it can not be used to generate dependency tables for the entire BN described in 5.3.2. If the structure of the BN generated from the Fly-In had fitted inside the existing BN model, it could replace parts of this model. Alas, this is not the case, and the dependency tables in the existing model are instead populated "by hand" and by using the method described in 5.4.1.

5.7 Testing

The model described in Section 5.3.2 consists of three layers. The testing of this model is carried out by testing each of these layers. The tests is done by instantiating nodes in a layer, registering the survivability, and then finding the combination of countermeasures that will give the highest increase in the survivability. The survivability is defined as the probability of the node `Survive` being in the *Survive* state. Testing is done using the HUGIN software, and whenever a node in the BN is instantiated the propagation of evidence is carried out automatically.

At first it is tested how changes in the first layer, i.e. changes to the states of the surrounding world, will affect the survivability found. Changes in the surrounding world is modelled by instantiating states in the Missile and RF source nodes. With |Missile| = 13 and |RF source| = 8 testing all combinations of states within these nodes requires a total of $13 \cdot 8 = 104$ tests. To make fewer tests, only combinations of 4 states for the Missile node and 3 states for the RF source node are tested. The set of appropriate countermeasures is found for each combination of states, and so are the increases in survivability when applying these countermeasures. The results of these tests are given in Table 5.11. Here the first two columns show the states initiated in the nodes mentioned, the third column shows the survivability before countermeasures are applied, the fourth column shows the best combination of countermeasures for the current threat scenario, and the last column has the survivability when the countermeasures have been applied. Countermeasures in the fourth column are only included if they make a significant difference in the survivability, and countermeasures improving the survivability with less than 0.01% are thus excluded. For all tests the nodes representing the availability of countermeasures are instantiated so that all countermeasures are available. The Manoeuvre node is also instantiated so that a manoeuvre will always be performed.

It can be noted that with this model the survivability is very close to 100% when no missile is launched. Launching a missile will give a large decrease in survivability, and while using appropriate countermeasures always increases the survivability, it will almost never get close to 100%. While these numbers may seem plausible the tests do show some inconsistencies in the model: An SA-2 missile is RF guided. Having the possible combination of the Missile being instantiated in the *SA-2* state while the RF source is instantiated in the *None* state shows that these nodes must be dependent on each other, and having them as being independent in the model is a flaw. Another flaw is that having the jammer turned on will never increase the survivability. While it is true that in some scenarios having the jammer turned off might increase the survivability, since it can then not attract missiles being guided towards jammer emissions,

Missile:	RF source:	P_S before:	Countermeasures:	P_S after:
No missile	None	100,00%	None	100,00%
No missile	X band	99,87%	Flares	99,99%
No missile	L band	99,90%	Flares	99,99%
SA-2	None	40,88%	Chaff	43,18%
SA-2	X band	64,97%	Chaff, no jammer	67,73%
SA-2	L band	64,80%	Chaff, no jammer	67,13%
Stinger Basic	None	58,15%	Flares	86,98%
Stinger Basic	X band	59,37%	Flares	91,09%
Stinger Basic	L band	57,67%	Chaff, flares	87,10%
HAWK	None	57,42%	Chaff	59,10%
HAWK	X band	65,96%	Chaff, no jammer	68,67%
HAWK	L band	68,63%	Chaff, no jammer	70,75%

Table 5.11: Testing the first layer of the BN model. Changing the surroundings (the nodes Missile and RF source) influence the survivability (P_S before). When countermeasures are applied the survivability changes (P_S after).

having it turned off will in general not have this effect. When RF radiation is present, and no missile has been launched, the use of a jammer will generally increase the survivability. This is not shown by the tests.

In testing the second layer of the model the influence on the survivability by input from on-board systems is evaluated. This is done with combinations of the states of the MWS, RWR, and Jammer nodes. Of the $|\mathsf{MWS}| \cdot |\mathsf{RWR}| \cdot |\mathsf{Jammer}| = 2 \cdot 4 \cdot 5 = 40$ combinations of states in these nodes $2 \cdot 3 \cdot 2 = 12$ are selected. The circumstances with these tests are the same as for testing the first layer. The results are shown in Table 5.12.

From these results it can be seen that having the RWR registering a radar lock will give a remarkably low survivability. Even if proper countermeasures are applied the survivability will not increase very much. The results also reveal other inexpediencies with the survivabilities related to RWR warnings: If the RWR detects a RF source, that is likely to be a threat, the survivability is higher than if the RWR detects no RF source. When prior to using countermeasures the survivability is at 100% no countermeasures can increase the survivability. For all other test cases the use of proper countermeasures will increase the survivability, although the size of this increase may be questionable.

Testing the first and the second layer shows the influence on changes in the survivability. Since survivability is the probability of the third layer node Survive being in the state *Survive*, tests of the third layer can show how changes to this node influence the probability distribution in the rest of the BN. To do this

MWS:	RWR:	Jammer:	P_S before:	Countermeasures:	P_S after:
No warning	No warnings	No RF waves	100,00%	None	100,00%
No warning	No warnings	Hostile RF waves - no jamming	100,00%	None	100,00%
No warning	No warnings	Hostile RF waves - jamming	100,00%	None	100,00%
No warning	RF source	No RF waves	100,00%	None	100,00%
No warning	RF source	Hostile RF waves - no jamming	100,00%	None	100,00%
No warning	RF source	Hostile RF waves - jamming	100,00%	None	100,00%
No warning	Lock	No RF waves	56,94%	Chaff, flares, jammer	96,01%
No warning	Lock	Hostile RF waves - no jamming	21,97%	Chaff, flares, jammer	48,34%
No warning	Lock	Hostile RF waves - jamming	19,97%	Chaff, flares	31,54%
Warning	No warnings	No RF waves	99,88%	Flares	99,96%
Warning	No warnings	Hostile RF waves - no jamming	80,26%	Flares	93,63%
Warning	No warnings	Hostile RF waves - jamming	81,80%	Flares	94,13%
Warning	RF source	No RF waves	99,99%	Flares	100,00%
Warning	RF source	Hostile RF waves - no jamming	99,98%	Flares	99,99%
Warning	RF source	Hostile RF waves - jamming	99,98%	Flares	99,99%
Warning	Lock	No RF waves	11,00%	Chaff, flares, no jammer	14,50%
Warning	Lock	Hostile RF waves - no jamming	8,74%	Chaff, flares, jammer	11,92%
Warning	Lock	Hostile RF waves - jamming	8,72%	Chaff, flares, jammer	11,81%

Table 5.12: Testing the second layer of the BN model. Changing the input from onboard systems (the nodes MWS, RWR and Jammer) influence the survivability (P_S before). When countermeasures are applied the survivability changes (P_S after). Notice that the P_S values are smaller in the last row, where the jammer is turned on, compared to the P_S values in the second to last row, where the jammer is off. This may be due to the fact that the jammer is often treated as a missile attractor, thus reducing the survivability when turned on.

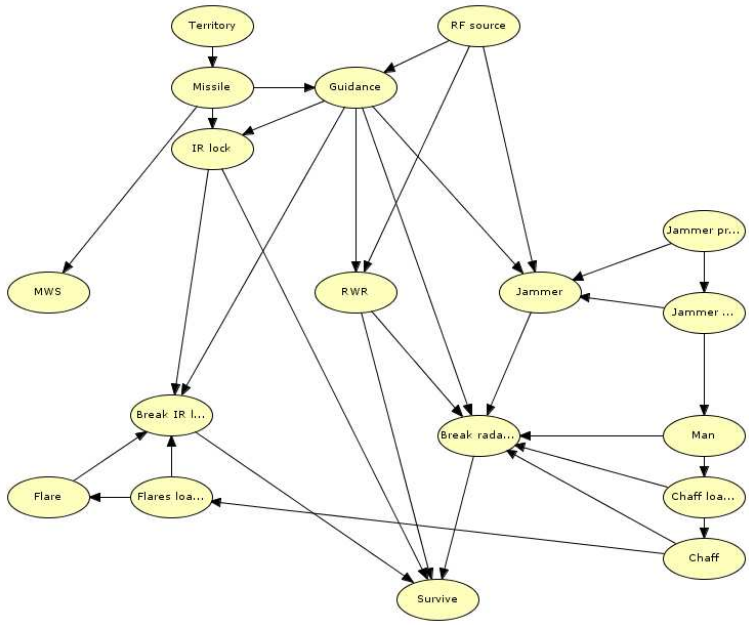


Figure 5.16: The BN with the decision nodes replaced by ordinary nodes. Picture exported from HUGIN

the decision nodes of the BN are converted into ordinary nodes with equal prior probabilities for all states in each node. This BN is illustrated in Figure 5.16.

This part of the test is performed using the combinations of nodes and states also used for testing the first and the second layer. At first the combinations of states from the Missile and RF source nodes, as used for testing the first layer, are tested again. Each test is performed by first instantiating the states in the two nodes. This results in a survivability similar to that given in Table 5.11 (P_S before). Instantiating the Survive node to the *Survive* state may result in changes in the probability distributions of the three countermeasures. For chaff and flares the probabilities of these being dispensed are registered, and for the jammer the mode having the highest probability is registered. If instantiating the Survive node suggests that either chaff or flares gets dispensed, or that the jammer is set in a given mode, the Chaff, Flares, and Jammer nodes are instantiated accordingly. With the countermeasure nodes being instantiated, the Survive node is un-instantiated, and the resulting survivability (P_S after) is registered. These results are given in Table 5.13

Converting the decision nodes in the BN into ordinary nodes has no effect on the

Missile:	RF source:	P_S before:	Chaff:	Flares:	Jammer mode:		P_S after:
No missile	None	100,00%	50,00%	50,00%	All	33,33%	100,00%
No missile	X band	99,87%	50,00%	50,06%	All	33,33%	99,99%
No missile	L band	99,90%	50,00%	50,04%	All	33,33%	99,99%
SA-2	None	40,88%	52,81%	50,00%	All	33,33%	43,18%
SA-2	X band	64,97%	51,88%	50,00%	Off	33,99%	67,73%
SA-2	L band	64,80%	51,62%	50,00%	Off	33,77%	67,13%
Stinger Basic	None	58,15%	50,00%	74,79%	All	33,33%	86,98%
Stinger Basic	X band	59,37%	50,00%	76,71%	All	33,33%	91,09%
Stinger Basic	L band	57,67%	50,00%	75,52%	All	33,33%	87,09%
HAWK	None	57,42%	51,47%	50,00%	All	33,33%	59,10%
HAWK	X band	65,96%	51,82%	50,00%	Off	33,97%	68,67%
HAWK	L band	68,63%	51,40%	50,00%	Off	33,72%	70,75%

Table 5.13: Testing the third layer of the BN model. The combinations of nodes and states are the same as for testing the first layer.

survivabilities found. This can be seen by comparing the values in the third (P_S before) and the last column (P_S after) with the similar columns in Table 5.11. The major difference between working with the BN containing decision nodes and the BN without decision nodes is that with the latter a good combination of countermeasures is found by instantiating the Survive node only.

In the final part of the BN test the combinations of states in the MWS, RWR, and Jammer nodes, as used for testing the second layer, is tested again. Table 5.14 holds the results of these tests.

Comparing the P_S values in Table 5.12 and Table 5.14 shows differences between the survivabilities found. The P_S values found before countermeasures are applied are identical, but after applying countermeasures they tend to be smaller in Table 5.14. The reason for this is that instantiating the Survive node does not guarantee that the best combination of countermeasures is found, and instantiating the countermeasures in the BN without action nodes, according to the set of countermeasures given in Table 5.12, will result in the same survivabilities as given here.

5.8 Discussion

It seems that a BN can be useful in modelling a domain where not all knowledge is categorical. In a DSS where decisions are based on uncertain observations the uncertainties can themselves, if they can be estimated, become part of the decision base. For the threat situation in a fighter aircraft the use of a BN seems adequate since decisions here will often be based on imperfect data from sensors on-board the aircraft. A reason for using a BN is that it is relatively easy to build a model for the threat response situation, and that this model can be built to reflect the uncertainties related to sensor output. The down side of using a BN for modelling the threat situation is that it requires a vast amount of knowledge to be represented in dependency tables in the BN.

Section 5.4.1 describes a semi-automatic method for populating dependency tables for nodes which depends on multiple parents. The method is not applicable for populating single parent dependency tables, and to be applicable for multiple parent dependency tables it needs a dependency table set up for each of the parents. The labour of finding proper values to populate these tables is not diminished by this method, and the resulting multi-dependency tables are not valid if the parents are interdependent.

Using the SL method described in Section 5.5 the dependency tables of the BN are

MWS:	RWR:	Jammer:	P_S before:	Chaff:	Flares:	Jammer mode:		P_S after:
No warning	No warnings	No RF waves	100,00%	50,00%	50,00%	All	33,33%	100,00%
No warning	No warnings	Hostile RF waves - no jamming	100,00%	50,00%	50,00%	Rx	50,04%	100,00%
No warning	No warnings	Hostile RF waves - jamming	100,00%	50,00%	50,00%	Auto	100,00%	100,00%
No warning	RF source	No RF waves	100,00%	50,00%	50,00%	Off	98,61%	100,00%
No warning	RF source	Hostile RF waves - no jamming	100,00%	50,00%	50,00%	Rx	50,09%	100,00%
No warning	RF source	Hostile RF waves - jamming	100,00%	50,00%	50,00%	Auto	100,00%	100,00%
No warning	Lock	No RF waves	56,94%	50,13%	84,23%	Off	38,68%	95,81%
No warning	Lock	Hostile RF waves - no jamming	21,97%	55,24%	74,69%	Rx	57,23%	28,84%
No warning	Lock	Hostile RF waves - jamming	19,97%	56,05%	73,08%	Auto	100,00%	31,54%
Warning	No warnings	No RF waves	99,88%	50,00%	50,04%	Off	33,35%	99,96%
Warning	No warnings	Hostile RF waves - no jamming	80,26%	50,00%	58,33%	Rx	50,07%	93,62%
Warning	No warnings	Hostile RF waves - jamming	81,80%	50,00%	57,53%	Auto	100,00%	94,13%
Warning	RF source	No RF waves	99,99%	50,00%	50,00%	Off	98,58%	99,99%
Warning	RF source	Hostile RF waves - no jamming	99,98%	50,00%	50,01%	Rx	51,70%	99,99%
Warning	RF source	Hostile RF waves - jamming	99,98%	50,00%	50,01%	Auto	100,00%	99,99%
Warning	Lock	No RF waves	11,00%	64,92%	50,75%	Off	33,53%	14,50%
Warning	Lock	Hostile RF waves - no jamming	8,74%	67,48%	50,31%	Rx	68,91%	11,82%
Warning	Lock	Hostile RF waves - jamming	8,72%	67,54%	50,20%	Auto	100,00%	11.81%

Table 5.14: Testing the third layer of the BN model. The combinations of nodes and states are the same as for testing the second layer.

populated regardless of the number of parents for each node. The drawback of using this method is that it requires a dataset faithfully representing the domain to model. If this dataset is available, either from real-world observations or from synthetic data, e.g. based on simulations, the SL feature will produce a usable BN or it will populate the dependency tables in an existing BN structure.

With both these methods the major part of the work is related to the gathering of detailed data describing the domain. A large part of the knowledge necessary for populating these dependency tables does either not exist or it is restricted and thus unavailable. Therefore constructing a BN to represent the relations in the EW domain for fighter pilots may become difficult or even impossible.

It is still possible to construct a simpler model that does not show the exact relations from a real world scenario. While this model will have flaws in some scenarios it may still be adequate for decision support in most situations.

In this work the HUGIN tool is used for both constructing the BN and for updating it during tests whenever nodes receive evidence. This is done using a graphical user interface, and while this seems fast it is difficult to evaluate whether the real-time requirement described in Section 3.3 is met. As described in Section 5.7 the model developed in this work has a small number of combinations of actions, and finding the combination yielding the best survivability can be done relatively fast. It is therefore assessed that the real-time requirement can be fulfilled using a BN for decision support if proprietary software for handling the necessary BN updates was written, so that time spent on e.g. updating a graphical user interface can be avoided.

5.9 Conclusion

This works has shown that it is possible to construct a simple BN to model the threat scenario and possible outcome of different combinations of countermeasures and manoeuvres for a fighter aircraft. The easy part of this is to define relationships between variables/nodes in the domain, while the hard part is to qualify these relationships by populating the dependency tables.

With a lack of real-world data the BN constructed is made on simple assumptions and it suffers from obvious errors, e.g. that the probabilities for survival will almost always be smaller than in a real-world scenario. Despite of this the tests show that most times the proper combination of countermeasures is found using the BN. It is not evident that this BN can be used in a DSS for fighter pilots, since the model constructed in this work is based on imaginary values only. In

constructing a model for real-world use more elaborate data should be used.

It seems that using a BN for decision support in fighter aircraft can be considered a viable approach only if necessary data describing the domain are available. If this is the case it is estimated that a proper BN can be constructed to be used for this purpose.

Chapter 6

The Mathematical Modelling Approach

When a threat occurs for which expendable countermeasures are an appropriate response these countermeasure may be applied to improve the survivability of the aircraft. As an aircraft is equipped with a limited number of expendables only, releasing all expendables at once may later bring the aircraft in a situation where expendables are needed without being available. Knowledge about threats that may engage the aircraft during a mission can help find the best overall use of the expendables. In situations where non-expandable countermeasures may offer nearly as good a protection as expendables, choosing the first over the latter may increase the survivability of the aircraft for the whole mission.

This chapter introduces a survivability measure. Determining which combination of countermeasures that will give the aircraft the highest probability of surviving a mission will be a matter of optimising this survivability measure. The influence on this measure by a set of countermeasures is described. A mathematical model to describe the countermeasures, temporal aspects about their use, and their contribution to the survivability is developed. Optimising the survivability for flights in a number of scenarios given is done by solving the problem described by the mathematical model for these flights.

6.1 Motivation

With the Prolog program (Chapter 4) or the BN (Chapter 5) the best countermeasure response can be found to the threat scenario at any given time. In finding these responses the countermeasures are assumed instantly active and the need for countermeasures at a later state in the mission is not considered. Introducing these aspects to the problem of finding the highest survivability for the aircraft may require the time frame of a mission to be discretised into a finite number of time steps. Finding the highest survivability for the pilot is then accomplished by finding the optimal combination of survivabilities for each of these time steps. To measure the survivability in each time step a survivability measure can be introduced. This measure must be based on both the current threat scenario and the countermeasures applied.

The problem of finding appropriate use of countermeasures during a mission will also include the amount of expendable countermeasures available and the restrictions imposed by e.g. the time it takes for the countermeasures to become active. For every feasible solution to this problem a survivability can be found, and the best solution will give the optimal survivability. For even very small problems described by few time steps only, evaluating all possible solutions to find the best will not be feasible due to the vast amount of solutions.

It will be possible to describe the countermeasures, how they are turned on, and when they become active, with a mathematical model. Also their influence on threats and the limitations set by the number of expendables available can be described. Therefore it is chosen to describe the problem using a mathematical model, which can then be solved to optimality.

6.2 Linear Programming

This section describes some basic theory on linear programming. Short introductions to both the CPLEX solver and to the General Algebraic Modeling System (GAMS) are also given. Readers familiar with these subject are encouraged to skip this section and continue with Section 6.3.

An optimisation problem is described using a number of equations and inequalities, and the optimal solution to the problem is the solution with the maximum/minimum value that satisfies the equations and inequalities describing the problem. The problem is described using a number of variables and parameters. Parameters are static values used to describe the problem, and the

variables are assigned values in the process of solving the problem. These values identify the problem solution. If the equations and inequalities contain only linear relations between the variables the model is called a *Linear Programming* model. If the solution is required to have only integer values for all variables involved *Integer Programming* is used. *Mixed Integer Programming* models are models where only a subset of the variables are required to be integer. Binary variables are special cases of integer variables.

To see the benefits of a mathematical model consider the task of preparing a fighter aircraft for a mission over enemy territory. The aircraft has a number of stations, and each of these stations may hold either a missile or a pylon with two canisters with flares. Let the two integer variables c and m describe the number of canisters and missiles on-board the aircraft, and let the parameter s be the number of stations on the aircraft. The relation between c, m, and s can be described by:

$$\frac{1}{2} \cdot c + m \leq s \tag{6.1}$$

The aircraft should be prepared for both engaging enemy aircraft and for missile attacks. Therefore, the inventory should contain at least three missiles and two canisters of flares. This is described by:

$$m \geq 3 \tag{6.2}$$
$$c \geq 2 \tag{6.3}$$

Let s be the number of stations on the aircraft. With seven stations, $s = 7$, the equations (6.1), (6.2), and (6.3) can be illustrated as the three lines in Figure 6.1. Here the triangle contains all the values of m and c that fulfil the three equations. Since m and c are required to be integer, only 16 combinations of m and c are valid.

Suppose the price of a missile is given by the parameter p_m, and the cost of filling a canister with flares is given by p_c. Let C be the total cost of equipping the aircraft for a mission. C is given by:

$$C = p_m \cdot m + p_c \cdot c \tag{6.4}$$

In finding the minimum cost for preparing the aircraft the equation (6.4) is called the *objective function* and the equations (6.1), (6.2), and (6.3) becomes *constraints*. The complete mathematical model is an integer program, and it is

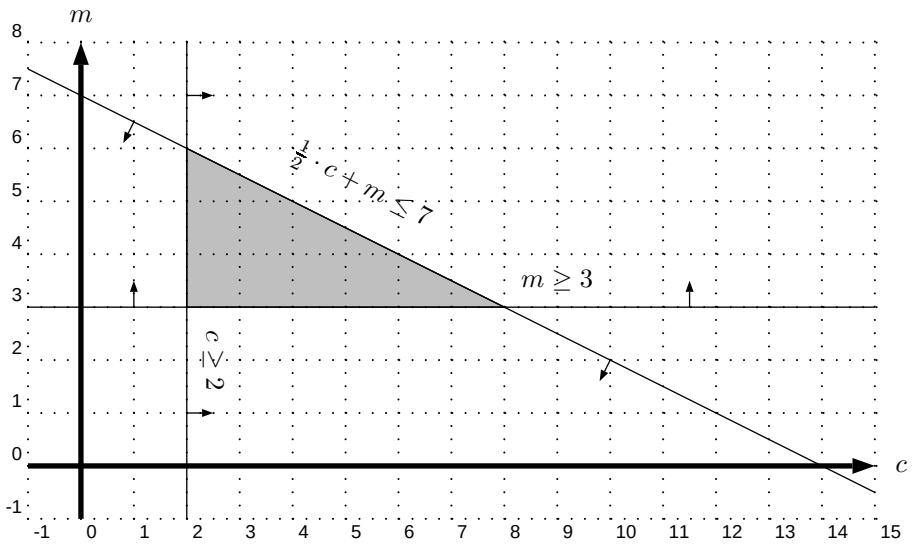


Figure 6.1: Three inequalities describing the need for flares and missiles in preparing an aircraft. The arrows describe where feasible solutions are found. All feasible solutions can be found inside the gray triangle.

given as:

Minimise

$$C = p_m \cdot m + p_c \cdot c$$

Subject to

$$\begin{aligned} \frac{1}{2} \cdot c + m &\leq s \\ m &\geq 3 \\ c &\geq 2 \\ m, c &\in \mathbb{N} \end{aligned}$$

With positive values for p_m and p_c the minimum cost is found in the lower left corner of the triangle in Figure 6.1, where $m = 3$ and $c = 2$.

6.2.1 CPLEX

The problem described by the mathematical model derived in Section 6.2 can be solved by visually inspecting the graph in Figure 6.1. For larger problems finding the solution by drawing a figure will not be feasible; if the model has more than two variables it may not even be possible to draw such a figure. Mathematical problems can be solved using solver software. CPLEX is such a solver, and it can be used to solve e.g. linear programs. More information on CPLEX can be found in Appendix E.

To make CPLEX solve a mathematical problem, the problem can be formulated in a file using the *lp-format* (*lp* for Linear Programming). Setting the prices for missiles and flares to $p_m = 1000$ and $p_c = 10$ respectively, the problem of finding the minimum cost for preparing the aircraft for a mission can be described by:

```
\ Finding the minimum cost when
\ preparing an aircraft for a mission
minimize
    1000m + 10c
subject to
    .5c + m <= 7
bounds
    m >= 3
    c >= 2
integer
    m c
end
```

Having CPLEX solve the problem generates the following results:

```
Integer optimal solution:  Objective =    3.0200000000e+03
Solution time =    0.00 sec.  Iterations = 0  Nodes = 0

Variable Name           Solution Value
m                             3.000000
c                             2.000000
```

The results state that having three missiles ($m = 3$) and two canisters of flares ($c = 2$) will give the minimum total cost of 3020. These results are the same as those found by inspecting the graphs in Figure 6.1.

6.2.2 GAMS

When problems get too large to be easily formulated using CPLEX, turning to GAMS may make the formulation possible. GAMS is a high-level modelling system for mathematical programming and optimisation. Problems may be written in the GAMS language, and when submitted to GAMS calculations in the program are performed, solvers such as CPLEX are invoked, and one or more output files are generated. GAMS is ideal for fast prototyping of large scale modelling applications, since the developer may focus mainly on the mathematical model. Implementing the problem in a program using a language such as C/C++, and linked to a solver library, may improve the running time of the program, while possibly increasing the time it takes to develop the model. In this work the combination of solving a mathematical model with GAMS using CPLEX is referred to as GAMS/CPLEX. The use of GAMS is described in [29]. More information on GAMS can be found in Appendix E.

The CPLEX model from Section 6.2.1 describes the problem of preparing an aircraft for a single mission. If more missions are planned, the constraints contained in the model will occur for each mission, and more constraints for the set of missions may appear. Instead of formulating every constraint for each mission a more general formulation can be made using GAMS.

Let M be the number of missions to be modelled. The inequalities (6.1), (6.2), and (6.3) will be repeated for every mission. This can be written by giving each of the variables a mission index i, $0 < i \leq M$:

$$\begin{aligned} \frac{1}{2} \cdot c_i + m_i &\leq s \\ m_i &\geq 3 \\ c_i &\geq 2 \end{aligned}$$

Assume that the number of missiles available is fixed to a value a. This can be described by:

$$\sum_{i=1}^{M} m_i \leq a.$$

Suppose the flares available are from an old batch. According to military procurement no new flares will be acquired before all flares from the old batch has been dispensed. There are enough old flares left to fill o canisters and all of

these must be dispensed during the set of missions modelled:

$$\sum_{i=1}^{M} c_i = o.$$

Next a GAMS program describing the requirements for preparing an aircraft for a number of missions is shown. Here $M = 5$, $a = 17$, and $o = 15$. While writing all constraints for the five missions in lp-format may use less line than the GAMS program, the GAMS program will remain the same size if the number of missions is changed to 50, 500, 5000, or more.

```
Sets
    mis         'Mission'   / 1 * 5 /
    ;
Parameters
    prMis       'The price of a missile'
    prCan       'The price of a canister full of flares'
    misAvail    'Number of missiles available'
    flAvail     'Number of canisters that must be filled'
    misReq      'Missiles required'
    canReq      'Canisters required'
    ;
    prMis       = 1000;
    prCan       = 10;
    misAvail    = 17;
    flAvail     = 15;
    misReq      = 3;
    canReq      = 2;
Variables
    totCost     The total cost
    m(mis)      Number of missiles for the i'th mission
    c(mis)      Number of canisters for the i'th mission
    ;
Scalars
    stations    Number of stations on the aircraft  / 7 /
    ;
Equations
    obj         Define objective function
    stat(mis)   Number of stations on the aircraft
    minMis(mis) Minimum number of missiles for a mission
    minCan(mis) Minimum number of canisters for a mission
    maxMis      Maximum number of missiles for all missions
```

```
    allFl       All flares left
    ;
obj         ..  totCost             =e=
                sum(mis, prMis*m(mis) + prCan*c(mis));
stat(mis)   ..  0.5*c(mis)+m(mis)   =l= stations;
minMis(mis) ..  m(mis)              =g= misReq;
minCan(mis) ..  c(mis)              =g= canReq;
maxMis      ..  sum(mis, m(mis))    =l= misAvail;
allFl       ..  sum(mis, c(mis))    =e= flAvail;

Model minCost /all/;
Solve minCost using mip minimizing totCost;

display m.L;
display c.L;
display totCost.L;
```

When solving the problem using GAMS/CPLEX the following is reported:

```
----     44 VARIABLE m.L  Number of missiles for the i'th mission
1 3.000,    2 3.000,    3 3.000,    4 3.000,    5 3.000
----     45 VARIABLE c.L  Number of canisters for the i'th mission
1 7.000,    2 2.000,    3 2.000,    4 2.000,    5 2.000
----     46 VARIABLE totCost.L  =    15150.000  The total cost
```

From here it can be seen that exactly three missiles are used for every mission. For the first mission the aircraft will be equipped with seven canisters of flares. These will occupy four stations, thus leaving three stations for the missiles.

6.3 The Framework

A mathematical model is developed for finding the optimal use of countermeasures during a flight. The model is described in Section 6.5. To test the model a number of flight descriptions are needed. For this a framework for designing test scenarios and generating flight descriptions is developed using MATLAB. This is done for two reasons: first of all data about real-world scenarios are difficult to obtain, and second, a threat scenario can be designed to test certain aspects of the model.

A scenario is described in a two-dimensional world. It consists of a number of

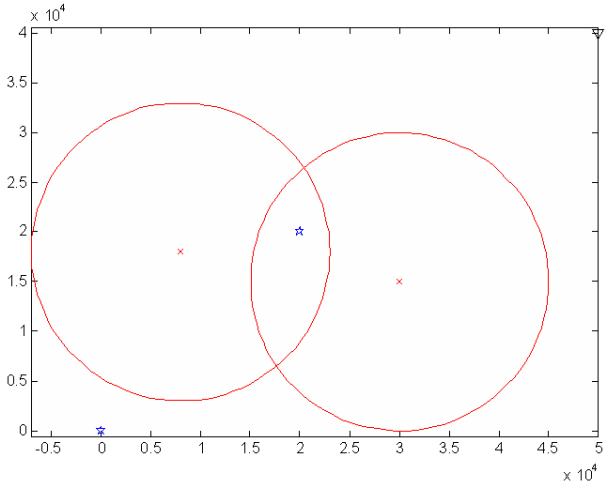


Figure 6.2: A scenario with two threats. The target is marked with an inversed triangle (▽), and each Intermediate Point (IP) is marked with a pentagram (☆). The target is placed in the upper right corner.

ground based radar threats, a target point, and a flight path given by a number of positions, each known as an Intermediate Point (IP). The aircraft flies in straight lines between the IPs in the order given. It is equipped with a number of countermeasures, some of which are limited in their number of deployments. Figure 6.2 shows a scenario with two threats and two IPs. The threats are shown as circles indicating the range of each threat. The units on the axes describe distances in metres.

Section 2.4 describes how ground based radar systems can have different modes, and that a radar system will change mode when an aircraft is detected. The ground based radars in this framework are simplified versions of real radar systems. A radar threat is here described by its location and its range/lethal envelope. Three types of radars are used, and they differ only by their ranges and their *lethalities*. The lethality describes how dangerous a threat is to the aircraft. More details on the lethality measure is given in Section 6.4.1. The lethality of a threat depends on the distance between the aircraft and the threat; the closer the aircraft is to the threat the larger the lethality. Outside the range of a threat the lethality is set to zero.

The pilot may use one of three countermeasures intended for RF threats only: jammer, towed decoy, and chaff. All countermeasures are simplified version of real-world countermeasures, and they are described next:

The jammer works by obfuscating any nearby threats. It works on a threat if the aircraft is within the range of this threat. The jammer antenna is positioned so it offers the biggest reductions if the threat is placed either in front of or behind the aircraft. When close to the threat the jammer offers no reduction in the threat lethality. The reduction increases as the aircraft gets further away from the threat, and it reaches its peak when the aircraft is at the rim of the range of the threat.

At any time the jammer will be in one of four states: off; on but not yet active; on and active; and active while turning off. In the model the jammer has to be in one of these states for a given period of time, i.e. for a given number of time steps, before its transition into the next state. No upper limits exist on the time the jammer should be off before it is turned on, or for how long it must be active before being turned off. Turning the jammer on and off takes a preset period of time, so a number of time steps is necessary on these transitional states. There is no limit to the number of times the jammer can be turned on. If it does not offer the best reductions it should not be turned on, since it may attract unnecessary attention from threats.

The towed decoy works basically as a "jammer on a string". One of the main differences between the on-board jammer and the towed decoy is the reductions given by the use of the decoy. Since the decoy is towed behind the aircraft it has the highest effect when the aircraft is flying away from the threats. It will be in one of four states: off; on but not yet active; on and active; and severed while the system is settling. It takes time for the decoy to become active since this requires the unreeling of a wire connected to the decoy. The towed decoy will stop jamming as soon as it has been turned off. It is assumed that the jammer may continue to jam as long as it is not severed. While the jammer may be turned on and off as often as necessary the number of towed decoys on-board the aircraft is limited. When it is turned off the wire is cut and the decoy is lost.

When chaff is dispensed it will form a cloud with a RCS comparable to that of the aircraft. Once formed, the cloud will maintain its RCS for a while. Chaff can be dispensed even if a chaff cloud is already formed. The frequency of dispensing chaff is limited by a latency period between two contiguous dispensings and by the amount of chaff available.

6.4 Optimise Survivability

The aim of this work is to give the aircraft the highest probability of surviving a mission as possible. Determining the probability of surviving, P_S, is not a trivial task, and a *survivability* measure, S, is introduced. No direct mapping between S and P_S is given, but the one is given by a monotonous bijection from the other.

Determining a value of S over a time frame can be done by subdividing the frame into a number of time steps, calculating the value for each of these steps, and finally combining the values into a single value for that time frame. This is described in the Sections 6.4.1 and 6.4.2. The optimal solution indicates when each of the available countermeasures must be activated. The usage is described for each of the time steps in the given time frame.

6.4.1 Lethality

The lethality experienced by the aircraft at any given time depends on the threats given in the current scenario. Let T denote the number of time steps in the time frame and let H be the number of threats in the current scenario. At the time t, $0 < t \leq T$, the lethality given by the threat h, $0 < h \leq H$ is defined by three factors: the *probability* P_{th} of a ground based radar actually posing a threat to the aircraft, the *threat lethality* $Ł_{th}$, and finally the current *lethality reduction* R_{th} for the threat which depends on e.g. the countermeasures in use.

Knowledge about the presence of a threat may be the result of intelligence reports or measurements done by on-board sensors. Both of these sources may be more or less reliable, and the probability of a radar system posing a threat may depend on the reliability of the source. A radar actually positioned in the scenario may also be out of function, thus not posing a threat.

The reduction of the lethality of threat h to the time t depends on two parameters: the angle towards the threat, α_{th} and the distance between the threat and the aircraft, ρ_{th} (see Figure 6.3). The distance between threat and aircraft is measured as a percentage of the threat range, and since a threat has no lethality outside its range the lethality is set to zero when $\rho_{th} > 100\%$. In Figure 6.4 the reductions of lethality for the three countermeasures are shown as functions of both angle and distance.

The aircraft will always point in the direction of flight, i.e. towards the next IP or the target. This is used when finding the angle between the aircraft

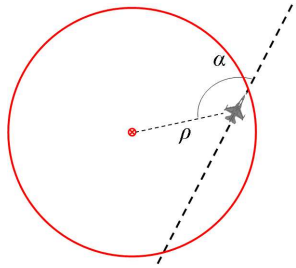


Figure 6.3: Angle α and distance ρ.

and a threat. The total reduction of threat lethality by a countermeasure is the product of the reductions from angle and distance. Reductions can not be added, and only the countermeasure with the highest reduction at any given time is considered as the countermeasure to use. Any synergy effects that may be obtained by combing available countermeasures are thus neglected. This is a deliberate choice as estimating the effect of multiple countermeasures mitigating a threat can then be avoided.

At any given time one of the countermeasures will be the one giving the best reduction of lethality. Since no countermeasure is needed when the aircraft is flying out of range of threats in the scenario a dummy countermeasure *NoCm* is introduced to be chosen here. Let $\mathcal{C}$ be the set of available countermeasures $\mathcal{C} = \{\text{Jammer}, \text{Decoy}, \text{Chaff}, \text{NoCm}\}$. The modes of all countermeasures are described by the vector $M_{\mathcal{C}t}$, e.g. $M_{\mathcal{C}t} =$ (off while active; on and active; off; off) which means that the jammer is off while still active, a towed decoy is both on and active, and both chaff and NoCm are off to the time t. The total *scenario lethality* experienced to the time t, Λ_t, is then given by:

$$\Lambda_t = \sum_{h=1}^{H} P_{th} \cdot L_{th} \cdot (1 - R_{th}^{\max}(\alpha_{th}, \rho_{th}, M_{\mathcal{C}t})) \tag{6.5}$$

where $R_{th}^{\max}(\alpha_{th}, \rho_{th}, M_{\mathcal{C}t})$ is the best reduction of the lethality of threat h to the time t offered by one of the countermeasures in $\mathcal{C}$. This reduction depends on the angle to the threat, α_{th}, the distance to the threat, ρ_{th}, and the modes, $M_{\mathcal{C}t}$, of the countermeasures available.

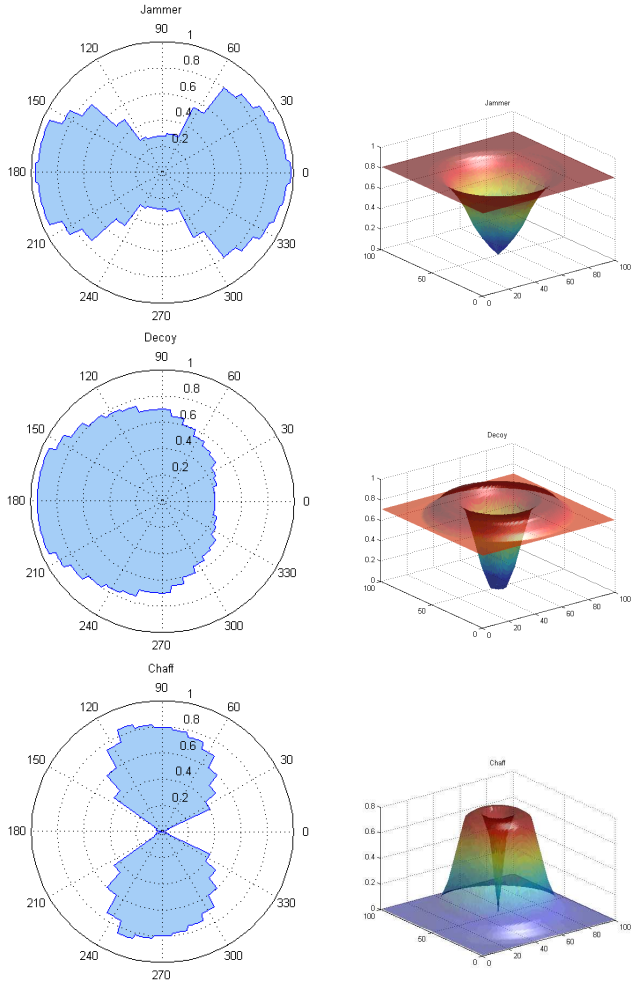


Figure 6.4: Countermeasure reductions. The polar plots at the left show the reductions as functions of the angle α, and the plots on the right shows the reductions as functions of the distance ρ to the threat. Here the threat is positioned in the middle of the plot at the point $(50, 50)$.

6.4.2 Survivability

The relation between the lethality and the survivability at time t, S_t is here defined as:

$$S_t = 1 - \Lambda_t \tag{6.6}$$

Since all factors in Λ_t have values between 0 and 1, both Λ_t and S_t will have values between 0 and 1. As stated earlier the survivability should not be mistaken for the probability of survival, as the scope of S_t suggests.

Finding an overall survivability measure by combining the survivabilities over time can be done in numerous ways. One way is by integrating the survivability over time:

$$S = \int_T S_t dt \tag{6.7}$$

Discretising the time frame into time steps, the combination of the survivabilities from each time step can be calculated as a sum:

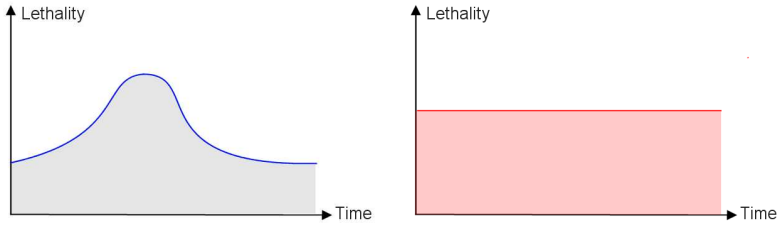
$$S_{\text{sum}} = \sum_{t=1}^{T} S_t \tag{6.8}$$

If the aircraft flies the entire time frame outside the range of any threat, the survivability in each time step is 1, and the sum of survivabilities then equals the number of time steps. Since this value will be more than one if more than one time step is used, the analogy to the P_S is lost. If S_{sum} is normalized by dividing it with the number of time steps in the time frame, it will have a value between 0 and 1, and the analogy to the probability of survival is regained:

$$S_{\text{sum,norm}} = \frac{1}{T} \sum_{t=1}^{T} S_t \tag{6.9}$$

Optimising S_{sum} may not ensure the aircraft the highest probability of surviving. In Figure 6.5 two courses with different development in scenario lethality are compared. The graph in Figure 6.5(a) shows varying scenario lethality as a function of time. This scenario lethality may be the result of the aircraft approaching a threat without any active countermeasures, which gives the rise in lethality. At some point in time the jammer may be turned on, which results in a decline in scenario lethality. In Figure 6.5(b) the scenario lethality is kept roughly constant, e.g. due to an active jammer. If the time steps are kept sufficiently small S_{sum} is comparable to the area under the graphs. The area in

6.5(a) is smaller than the area in 6.5(b), indicating that the first solution is the best. The peaking lethality means that the survivability at this point reaches a low, giving a low probability of surviving. Having low lethality for the rest of the mission is of little importance to the pilot, if the momentary low probability of surviving means that he will not survive the peak in lethality.



(a) Varying lethality. (b) Constant lethality.

Figure 6.5: Two courses with different development in lethality.

A good solution to finding the best use of countermeasures over time should avoid time steps with low survivability. This suggests another survivability measure where the object is to maximise the lowest survivability for a time step during the time frame:

$$S_{\text{min}} = \min_{t \leq T} S_t \tag{6.10}$$

Maximising the S_{min} ensures that the value of the lowest survivability over time will be as high as possible. Since a low survivability may be unavoidable in some scenarios, maximising S_{min} will not necessarily choose the solution with the best survivability for all other time steps than the one with the lowest survivability.

Although S_t should not be mistaken for P_S, it may in some ways be treated like a probability measure. Since the probability of surviving any time steps past a given time step, t_0, depends on surviving all prior time steps, including t_0, the survivability can be given as the product of survivabilities for each time step:

$$S_{\text{prod}} = \prod_{t=1}^{T} S_t \tag{6.11}$$

As S_t has a value between 0 and 1 so has S_{prod}. Interpreting S_t as a probability makes S_{prod} the probability of surviving the entire time frame.

It is the subjective assessment of the author that S_{prod} is the best survivability measure introduced. Despite of this S_{sum} and $S_{\text{sum,norm}}$ are the survivability measures chosen for further work. These are chosen since it is assumed that calculating the survivability using addition only, as in both S_{sum} and $S_{\text{sum,norm}}$,

will be slightly faster than using multiplication as in S_{prod}. The survivability measure chosen for the mathematical model will be the same as the one used by the metaheuristics in Chapter 7. Here the time it takes to calculate the survivability may have a substantial effect on the results found.

6.4.3 Distribution of Time Steps

The time frame for the most critical part of a mission will usually be no more than three to five minutes. To find the survivability within this amount of time the time frame is divided into a number of time steps. The route for the mission is sampled with this number of time steps, and the position of the aircraft is found for each time step. From these positions the distances and angles to threats in the scenario are found, and the survivability is calculated. Figure 6.6 shows a flight with 19 sampled positions.

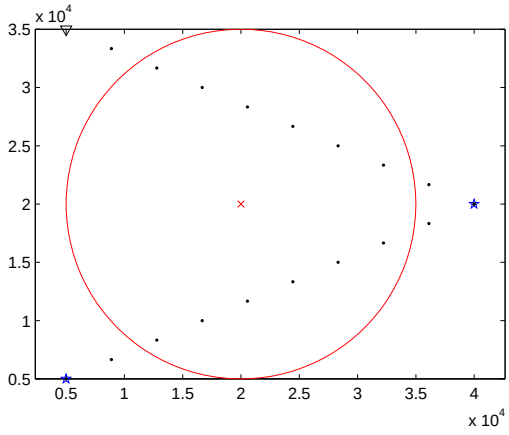


Figure 6.6: 19 positions on a simple route.

It is a requirement that the system can deliver solutions to changes that has occurred within the last 200 milliseconds. It may therefore seem reasonable to suggest the length of each time step to be no more than 200 milliseconds. For each minute of flight this will add 300 time steps to the time frame. Assuming that the aircraft will be within the range of threats for no more than five minutes during a mission, the total number of time steps within the time frame will not be more than 1500. According to tests described in Section 6.7 the survivability found for a flight may not vary much if this number is reduced.

The time steps in this work are distributed evenly over the time frame. While this makes the sampling of location points easy it may not be the best way to distribute the time steps. If the aircraft is within the lethal range of a threat the pilot must focus on this threat for more reasons: if he does not survive the encounter with this threat he will not survive encounters with future threats either, and the probability of this threat actually posing a threat is likely to be higher than for any threats that may or may not be encountered later on. Focussing on threats in the near future may be reflected in the distribution of time steps. This can be done by sampling the first period of time, e.g. the next 30 seconds, with a higher frequency, the next couple of minutes with a lower frequency, and finally having only a few samples in the last part of the time frame.

Keeping the focus on the actions in the near future can also be done by weighting contributions from early time steps higher in the survivability measure. This can e.g. be done in the estimation of the probability for each threat. Like with the unevenly distribution mentioned before meeting threats in the far future will have only minor influence on the survivability found. The down side to this approach is that the benefit from optimising the use of expendable countermeasures may be reduced. This has to be considered when introducing either uneven distribution or weighing of time steps.

6.4.4 Deployment Scheme

A solution to the problem of finding the best use of countermeasures takes the form of a *deployment scheme*. In a deployment scheme two columns describe each of the countermeasures. The one column indicates the time steps in where the countermeasure is active and the other column describes when the countermeasure is turned on/dispensed. Figure 6.7 shows such a deployment scheme.

When a countermeasure is active it will remain so for a number of time steps. Each of these collections of time steps is called a *deployment interval*. For the towed decoy the maximum number of deployment intervals equals the number of decoys on-board the aircraft. If the towed decoy is not needed for a short period of time during flight, it may be necessary to keep it active to avoid using more decoys than are available. For chaff a deployment interval can consist of more chaff dispensings.

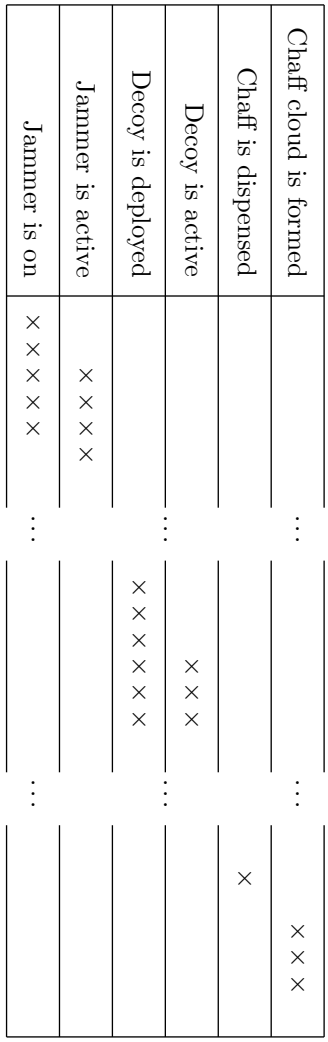


Figure 6.7: A deployment scheme showing the status of each of the countermeasures at every time step. Here the countermeasures are active one at a time. They can be active simultaneously, e.g. if they counter different threats.

6.5 Modelling the Problem

$S_{\text{sum,norm}}$ is chosen as the survivability measure to optimise in the mathematical model. This is chosen over S_{sum} since comparing results for flights having different number of time steps is more difficult using the latter. For simplicity P_{th} is set to 100% during the time steps where the aircraft is within the range of the threat h, while it is set to 0% when flying outside the range of the threat. This in effect reduces the need for estimating P_{th} in finding the lethality. Values for the threat lethalities and the lethality reductions as functions of both ρ and α are found in look-up tables. These tables are constructed for testing purposes only, and they do not necessarily reflect any real-world behaviour.

Parameters used in the model can be divided into scenario related and independent parameters. The scenario related parameters describe the distances and angles between the aircraft and each of the threats at any time step. The scenario independent parameters describe the general reductions by any of the countermeasures as functions of both distance and angle.

6.5.1 General Constraints

The time frame contains T time steps, and the scenario describes H threats. The survivability to maximize is $S_{\text{sum,norm}}$ as defined in (6.9). From the survivability a penalty term $\mathcal{P}$ is subtracted. This penalty is introduced to ensure that all countermeasures are on and active for the necessary time steps only. The following sections describe the penalties included in $\mathcal{P}$. Every penalty in $\mathcal{P}$ is weighted with a small value ε. This is chosen so that the difference between the survivability and the objective function is relatively small even when a large number of penalties are included in the latter. The objective function is given as:

$$\max Z = S_{\text{sum,norm}} - \mathcal{P} \tag{6.12}$$

The survivability $S_{\text{sum,norm}}$ is given by:

$$S_{\text{sum,norm}} = \frac{1}{T} \sum_{t=1}^{T} (1 - \Lambda_t) \tag{6.13}$$

The scenario lethality in the time step t, Λ_t, is defined by:

$$\Lambda_t = \sum_{h=1}^{H} P_{th} \cdot L_{th} \cdot (1 - R_{th}^{\max}(\alpha_{th}, \rho_{th}, M_{\mathcal{C}t})) \tag{6.14}$$

The variable $R_{th}^{\max}$ is the value of the reduction of lethality of the threat h to the time step t. It takes the highest possible value of the reduction of one of the countermeasures. R_{th}^{J} is the reduction from the jammer, R_{th}^{D} is the reduction from the decoy, and R_{th}^{C} is the reduction from chaff. The values of these are calculated from the distances and angles to the threats in the scenario. Since other constraints may prevent the countermeasure with the highest reduction from being active, simply assigning the highest value of R_{th}^{J}, R_{th}^{D}, and R_{th}^{C} may not be correct.

Let the binary variables A_t^J, A_t^D and A_t^C describe if the countermeasures are active to the time t. The variables describe the states of the jammer, the towed decoy, and chaff respectively. With these variables the value of $R_{th}^{\max}$ is calculated as:

$$R_{th}^{\max} = \max\{R_{th}^{J}A_t^J, R_{th}^{D}A_t^D, R_{th}^{C}A_t^C\} \tag{6.15}$$

Expressing (6.15) using inequalities gives the following constraints:

$$\begin{aligned}
R_{th}^{\max} &\leq R_{th}^{J}A_t^J + M(1 - a_{th}^{J}) && (6.16)\\
R_{th}^{\max} &\leq R_{th}^{D}A_t^D + M(1 - a_{th}^{D}) && (6.17)\\
R_{th}^{\max} &\leq R_{th}^{C}A_t^C + M(1 - a_{th}^{C}) && (6.18)\\
R_{th}^{\max} &\leq M(1 - a_{th}^{N}) && (6.19)
\end{aligned}$$

Here a_{th}^{cm} is a binary variable describing the use of the countermeasures. The superscript on a indicates the countermeasure (J = jammer, D = decoy, C = chaff, and $N = NoCM$). It has the value 1 if cm is the active countermeasure yielding the highest reduction of the lethality of threat h to the time t, and 0 otherwise.

For each threat h, only one countermeasure is reducing the lethality to the time t. To ensure that exactly one countermeasure is assigned to mitigate each threat, the following constraint is introduced:

$$\sum_{cm \in \mathcal{C}} a_{th}^{cm} = 1 \tag{6.20}$$

When flying outside the range of a threat no real countermeasure will be deployed, i.e. the dummy countermeasure $NoCM$ must become active. Let ρ_{th} describe the distance between the aircraft and the threat h to the time t. ρ_{th} has a value less than or equal to 1 when the aircraft flies within the range of the threat, and it is more than 1 when flying outside the range. The parameter ρ_{th}^{*} is introduced as a distance that can not have a value greater than 2: $\rho_{th}^{*} = \min(\rho_{th}, 2)$. Assigning a value to the binary variable a_{th}^{N}, indicating when

no countermeasure should be deployed, is done by the constraints:

$$a_{th}^{N} \leq \rho_{th} \tag{6.21}$$
$$a_{th}^{N} \geq \rho_{th}^{*} - 1 \tag{6.22}$$

6.5.2 Generic Time-related Constraints

After a countermeasure has been turned on, it takes a while for it to become active. The state of a countermeasure is described with at least two decision variables for each time step, one indicating if the countermeasure is turned on, and one indicating if it is active. Most of the constraints in the model describe the relations between these two variables.

The need for a countermeasure being active may be expressed in non-contiguous time steps, as illustrated in Figure 6.8.

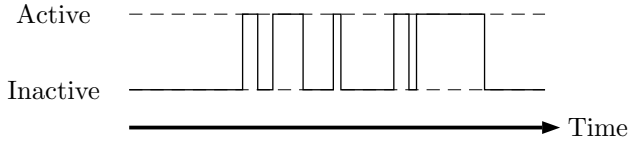


Figure 6.8: Example of the need for a countermeasure being active. The need is defined by the presence of threats in the scenario.

Some countermeasures will not be active to a given time unless they are turned on some time in advance, and they may need to be active for another period of time while turning off. It can therefore be necessary to keep the countermeasure active for longer time than it is required by the scenario.

Figure 6.9 shows the relations between two binary variables, O_t and A_t, for a given countermeasure to the time t. O_t has the value 0 when the countermeasure is turned off, and the value 1 when it is turned on. The variable A_t indicates if the countermeasure is active, $A_t = 1$, or not, $A_t = 0$, in the time step t. The time it takes from a countermeasure is turned on at $t = t_0$, $O_{t_0} = 1$, to it becomes active at $t = t_1$, $A_{t_1} = 1$, is given by the number of time steps $T_{\text{start}} = t_1 - t_0$. When it is again turned off, $O_{t_2} = 0$, it takes another period of time, T_{end}, before it stops being active at t_3.

The variables describing the status of a countermeasure should be 0 unless explicitly set to 1. This is to ensure that a countermeasure is not turned on, or being active, unless there is a need for it. It is ensured by subtracting a penalty

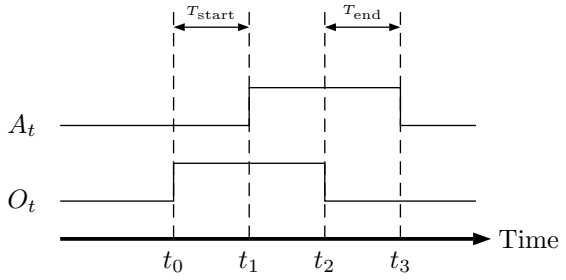


Figure 6.9: The binary variables describing when a countermeasure is turned on ($O = 1$), and when it is active ($A = 1$).

value from the objective function whenever the value of one of these variables is set to 1.

In general the deployment of a countermeasure can be described in five phases, as shown in Figure 6.10. In the first phase the countermeasure is not turned on, and it is thus not active either. The next phase describes the time that goes from the countermeasure is turned on and until it becomes active. This phase has a fixed duration that depends on the countermeasure. The situation where the countermeasure is both turned on and being active is given in the third phase. The duration of this phase depends on the need for the countermeasure being active, and it is determined by the presence of threats. Usually this phase will endure for a period much longer than that of the second phase. In the fourth phase the countermeasure has been turned off, and it continues to be active for a fixed period of time. The fifth phase describes again the situation where the countermeasure is neither turned on or active. This phase is also the first phase of the next deployment of the countermeasure. Of the three countermeasures described here only the jammer will follow all of the five phases.

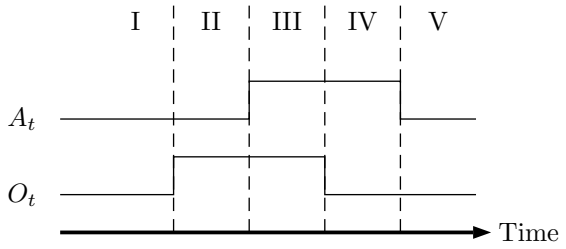


Figure 6.10: The five phases of a countermeasure deployment.

The binary variable O'_t is introduced to describes when the countermeasure gets turned on, i.e. when the second phase of the deployment commences. The variable is assigned the value 1 exactly in the time step where the countermeasure gets turned on, and 0 in all other time steps. To find the number of times the countermeasure has been turned on during an entire mission, one needs only to count the number of time steps where $O'_t = 1$.

The binary variable O''_t is introduced to describe when the countermeasure is again turned off, i.e. when the fourth phase begins. Here O''_t is 1 in the time step where the countermeasure gets turned off, and 0 otherwise. The relations between the variables O_t, O'_t, and O''_t are illustrated in Figure 6.11.

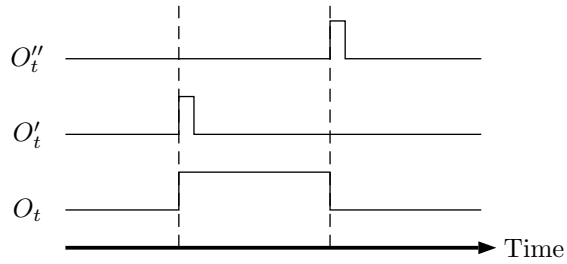


Figure 6.11: The binary variables O'_t and O''_t describe when the countermeasure is turned on and off.

The value of O'_t is 1 exactly when the countermeasure is turned on. The countermeasure must be turned on T_{start} time steps before becoming active, i.e. $A_{t+T_{\text{start}}} = 1$ and $A_{t+T_{\text{start}}-1} = 0$. This can be described by the constraints:

$$O'_t \geq A_{t+T_{\text{start}}} - A_{t+T_{\text{start}}-1} \tag{6.23}$$

$$O'_t \geq 0 \tag{6.24}$$

Since O'_t is a binary variable, and it can have the values 0 and 1 only, the constraint in (6.24) is not needed. To ensure that the value of O'_t is 0 during all remaining time steps, a penalty value is subtracted from the objective function whenever the value of O'_t is 1, as with A_t and O_t.

The value of O''_t can be found using the constraint:

$$O''_t \geq O_{t-1} - O_t \tag{6.25}$$

This will ensure that O''_t has the value 1 whenever the countermeasure gets turned off. Subtracting a penalty value from the objective function whenever the value of O''_t is 1 will again limit the number of time steps where the value of O''_t is 1.

During the first phase the values of O_t and A_t are set to 0 by their inclusion in the objective function. In the second phase it must be ensured that the value of O_t is set to 1 for all values of t within this phase. This can be formulated as:

$$\begin{gathered} O'_t = 1 \Rightarrow O_t = \ldots = O_{t+T_{\text{start}}-1} = 1 \\ \Updownarrow \\ O'_t = 1 \Rightarrow \sum_{j=t}^{t+T_{\text{start}}-1} O_j = T_{\text{start}} \\ \Updownarrow \\ \sum_{j=t}^{t+T_{\text{start}}-1} O_j \geq T_{\text{start}} \cdot O'_t \end{gathered} \tag{6.26}$$

It is not possible to turn off the countermeasure before it has been turned on for at least T_{start} time steps. Therefore it is not possible to assign a value of 1 to O''_t for any t within the second phase. Since the value of O_t can not become 0 within this phase, this requirement is implicitly fulfilled.

In the third phase of deployment the countermeasure should be both turned on and active. The start of this phase can be found by counting the number of time steps the countermeasure has been turned on, and set A_t to 1 when this count exceeds T_{start}. Let C_t be an integer variable that is incremented with 1 for each time step for as long as the countermeasure is on, $O_t = 1$, starting from 0. When the countermeasure is turned off again, the value of C_t returns to 0. The relations between the variables O_t, A_t, and C_t are illustrated in Figure 6.12.

With the introduction of C_t the value of A_t can be set by the constraint:

$$T_{\max} \cdot A_t \geq C_t - T_{\text{start}} \tag{6.27}$$

Here $T_{\max}$ is a big number that ensures that the left-hand side of the inequality is greater than the right-hand side whenever the C_t is greater than T_{start}. It describes the maximum number of time steps the countermeasure can be on, and since it may be so for the entire mission, $T_{\max}$ can be set to the number of time steps T in the problem description, i.e. $T_{\max} = T$.

C_t is defined by the following constraints, which are almost similar to constraints described in [9]:

$$C_t \leq C_{t-1} + 1 \tag{6.28}$$

$$C_t + T_{\max}(1 - O_t) \geq C_{t-1} + 1 \tag{6.29}$$

$$C_t - T_{\max} \cdot O_t \leq 0 \tag{6.30}$$

$$C_t \geq 0 \tag{6.31}$$

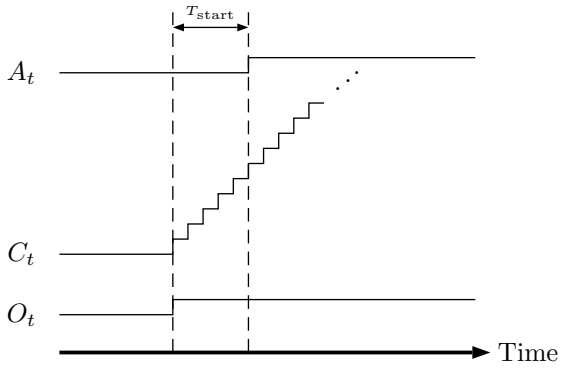


Figure 6.12: Relations between the variables O_t, A_t, and C_t during the second and third phase.

The constraints (6.28) and (6.29) will increment C_t with 1 for each time step as long as the countermeasure is on. C_t must have the value 0 when the countermeasure is turned off, and this is ensured by the constraints (6.30) and (6.31).

To ensure that the countermeasure is kept turned on for the duration of the third phase the value of O''_t can be related to the time where the countermeasure is no longer active. This is done by:

$$A_{t+T_{end}} \leq 1 - O''_t \tag{6.32}$$

The fourth phase, where the countermeasure is active for T_{end} time steps while it has been turned off, bear some similarity to the second phase. The phase is started when the countermeasure gets turned off, i.e. $O''_t = 1$, and it can be described by the constraint:

$$\sum_{j=t}^{t+T_{\text{end}}-1} A_j \geq T_{\text{end}} \cdot O''_t \tag{6.33}$$

As it is not possible to turn off the countermeasure during the second phase, it is also not possible to turn it on again during the fourth phase. Since constraint (6.33) does not include O_t this has to be stated explicitly. This is done by:

$$\sum_{j=t}^{t+T_{\text{end}}-1} O_j \leq T_{\text{end}} \cdot (1 - O''_t) \tag{6.34}$$

If T_{start} and T_{end} is of the same size for a given countermeasure, all relations between O_t and A_t may be described using a single constraint:

$$A_t = O_{t-T_{\text{start}}} \tag{6.35}$$

Since T_{start} and T_{end} will often have the magnitude of a few seconds, while the countermeasure may be active for several minutes, assuming equality between T_{start} and T_{end} will introduce only a minor error in the model. The use of a single constraint such as (6.35) is likely to decrease the computation time for finding the optimal solution, and it is thus beneficial to use this instead. The original formulation of dependencies between A_t and O_t will ensure that if a countermeasure can not be turned both off and on in between deployment intervals, it will remain on. The constraint in (6.35) will not insure this. On the contrary, gaps in A_t will be repeated in O_t.

6.5.3 Jammer-related Constraints

The status of the jammer can be described using the five phases mentioned above. The binary variable telling if the jammer is turned on or off to the time t is called O_t^J, and whether it is active or not is given by the binary variable A_t^J. The time it takes for the jammer to become active after it has been turned on is called T_{JA}, and the number of time steps it will remain active after having been shut down is called T_{JS}. Relations between the variables O_t^J and A_t^J, and the parameters T_{JA} and T_{JS} are illustrated in Figure 6.13.

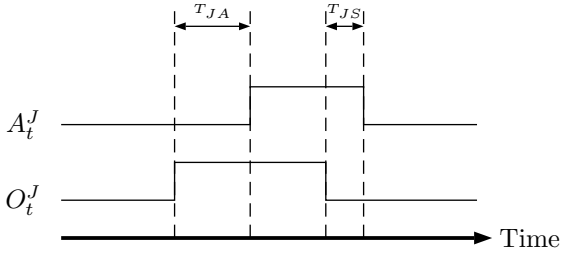


Figure 6.13: The relations between turning the jammer on/off and it being active.

The binary variables $O_t'^J$ and $O_t''^J$ are introduced to indicate when the jammer

is turned on and off, respectively. They are defined by the constraints:

$$O'^J_t \geq A^J_{t+T_{JA}} - A^J_{t+T_{JA}-1} \tag{6.36}$$
$$O''^J_t \geq O^J_{t-1} - O^J_t \tag{6.37}$$

Both O^J_t and A^J_t are included in the objective function, and their values during the first phase are thus set to 0. In the second phase the jammer has been turned on and it can not be turned off for T_{JA} time steps. This is controlled by the constraint:

$$\sum_{j=t}^{t+T_{JA}-1} O^J_j \geq T_{JA} \cdot O'^J_t \tag{6.38}$$

In the third phase the value of A^J_t is controlled by the integer variable C^J_t which counts the number of time steps since the jammer was turned on. Keeping the countermeasure turned on, the definition of C^J_t, and the relation between C^J_t and A^J_t are given by the constraints:

$$C^J_t \leq C^J_{t-1} + 1 \tag{6.39}$$
$$C^J_t + T(1 - O^J_t) \geq C^J_{t-1} + 1 \tag{6.40}$$
$$C^J_t - T \cdot O^J_t \leq 0 \tag{6.41}$$
$$C^J_t \geq 0 \tag{6.42}$$
$$T \cdot A^J_t \geq C^J_t - T_{JA} \tag{6.43}$$
$$A^J_{t+T_{JS}} \leq 1 - O''^J_t \tag{6.44}$$
$$\tag{6.45}$$

The fourth phase begins when the jammer is shut down. This phase endures for T_{JS} time steps during which the jammer will remain active, $A^J_t = 1$, while it can not be turned on again, $O^J_t = 0$. This is described by the constraints:

$$\sum_{j=t}^{t+T_{JS}-1} A^J_j \geq T_{JS} \cdot O''^J_t \tag{6.46}$$
$$\sum_{j=t}^{t+T_{JS}-1} O^J_j \leq T_{JS} \cdot (1 - O''^J_t) \tag{6.47}$$

6.5.4 Decoy-related Constraints

As with the jammer the towed decoy will become active some time after it has been deployed. A difference between the jammer and the decoy is, that the

decoy stops working as soon as it is severed. Although no towed decoy is active, the system controlling the decoys needs to settle for a period of time before the next decoy can be deployed. The variables for describing the status of the towed decoy are: O_t^D (is the decoy deployed), A_t^D (is the decoy active), $O_t'^D$ (the decoy gets deployed), $O_t''^D$ (the decoy gets severed), and C_t^D (for how long has the decoy been deployed). Figure 6.14 illustrates the relations between the variables and the time parameters: T_{DA}, the time it takes for a deployed decoy to become active, and T_{DR}, the time it takes for the system to settle when a decoy has been released.

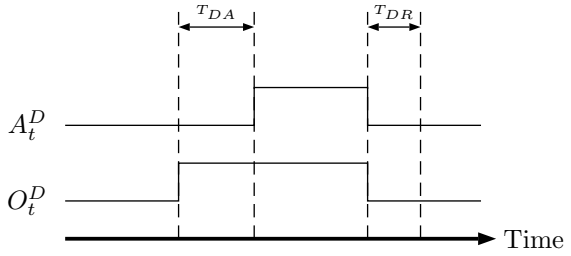


Figure 6.14: The relations between deploying the towed decoy and it being active.

For the first three phases of the deployment most constraints for the towed decoy are similar to the constraints for the jammer. The main differences are that O_t^J is replaced by O_t^D, A_t^J by A_t^D, etc.

$$O_t'^D \geq A^D_{t+T_{DA}} - A^D_{t+T_{DA}-1} \tag{6.48}$$

$$O_t''^D \geq A^D_{t-1} - A^D_t \tag{6.49}$$

$$\sum_{j=t}^{t+T_{DA}-1} O_j^D \geq T_{DA} \cdot O_t'^D \tag{6.50}$$

$$C_t^D \leq C^D_{t-1} + 1 \tag{6.51}$$

$$C_t^D + T(1 - O_t^D) \geq C^D_{t-1} + 1 \tag{6.52}$$

$$C_t^D - T \cdot O_t^D \leq 0 \tag{6.53}$$

$$C_t^D \geq 0 \tag{6.54}$$

$$T \cdot A_t^D \geq C_t^D - T_{DA} \tag{6.55}$$

$$O_t^D \geq A_t^D \tag{6.56}$$

For the towed decoy the fourth phase begins when the decoy is released. During the T_{DR} time steps of this phase the released decoy is not active, and no new

decoy can be deployed. Having a penalty value subtracted from the objective function for each time step t in which $A_t^D = 1$ will ensure that the decoy is not set as active during this phase. That no new decoy gets deployed is ensured by:

$$\sum_{j=t}^{t+T_{DR}-1} O_j^D \leq T_{DR} \cdot (1 - O_t''^D) \tag{6.57}$$

A maximum of K_D towed decoys can be deployed during the mission. Since $O_t'^D$ has the value 1 each time a decoy gets deployed, this is given by:

$$\sum_{t=1}^{T} O_t'^D \leq K_D \tag{6.58}$$

6.5.5 Chaff-related Constraints

The dispensing of chaff will not follow the five phases of countermeasure deployment previously presented. Dispensing of chaff is initiated during a single time step, and after a short period of time a chaff cloud is formed behind the aircraft. This cloud will last for another period of time before being dissipated. After dispensing chaff the next dispense may take place after a short latency period, regardless of whether the first cloud is yet formed, or if the effect of it has gone. The binary variables O_t^C and A_t^C describes the use of chaff to the time t. O_t^C is 1 only during the time steps where chaff dispensing is initiated, and A_t^C is 1 when a chaff cloud is formed and potentially having an effect.

Three parameters describe the periods of time involved in chaff dispensing. T_{CF} is the time it takes for a chaff cloud to be formed after chaff dispensing is initiated, T_{CD} denotes the duration of a chaff cloud, before it is dissipated, and T_{CL} is the latency between contiguous chaff dispensings. The relations between these parameters and the variables O_t^C and A_t^C are illustrated in Figure 6.15

When chaff is dispensed, the chaff cloud will be formed after T_{CA} time steps, and it will dissipate after another T_{CD} time steps. This can be expressed as:

$$O_t^C = 1 \Rightarrow A_{t+T_{CF}}^C = \ldots = A_{t+T_{CF}+T_{CD}-1}^C = 1$$

$$\Updownarrow$$

$$O_t^C = 1 \Rightarrow \sum_{j=t+T_{CF}}^{t+T_{CF}+T_{CD}-1} A_j^C = T_{CD}$$

$$\Updownarrow$$

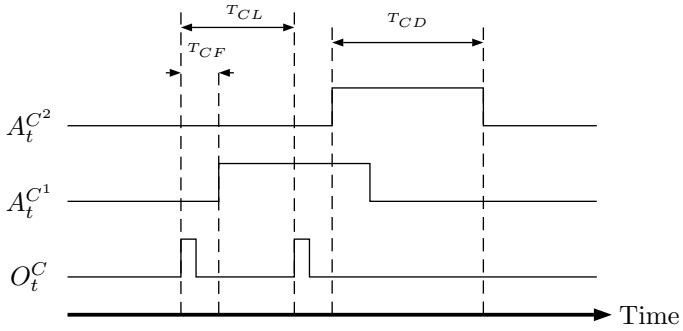


Figure 6.15: The relations between dispensing chaff and a chaff cloud being formed. $A_t^{C^1}$ and $A_t^{C^3}$ are set to 1 when the cloud is formed after the first and second dispensing respectively. The value of A_t^C is 1 if either $A_t^{C^1}$ or $A_t^{C^2}$ is 1.

$$\sum_{j=t+T_{CF}}^{t+T_{CF}+T_{CD}-1} A_j^C \geq T_{CD} \cdot O_t^C \tag{6.59}$$

If no chaff has been dispensed for a while, a chaff cloud is no longer formed:

$$O_{t-T_{CD}-T_{CF}+1}^C = 0 \wedge \ldots \wedge O_{t-T_{CF}}^C = 0 \Rightarrow A_t^C = 0$$

$$\Updownarrow$$

$$A_t^C \leq \sum_{j=t-T_{CD}-T_{CF}+1}^{t-T_{CF}} O_j^C \tag{6.60}$$

No chaff dispensing can be initiated for T_{CL} steps after the previous dispensing has taken place:

$$O_t^C = 1 \Rightarrow O_{t+1}^C = \ldots = O_{t+T_{CL}}^C = 0$$

$$\Updownarrow$$

$$O_t^C = 1 \Rightarrow \sum_{j=t+1}^{t+T_{CL}} O_j^C = 0$$

$$\Updownarrow$$

$$\sum_{j=t+1}^{t+T_{CL}} O_j^C \leq T_{CL}(1 - O_t^C) \tag{6.61}$$

As with the towed decoy, chaff can only be deployed a limited number of times. For chaff this number is K_C. Chaff dispensing is initiated during a single time step, and the number of time steps in where a dispensing is initiated may not exceed K_C.

$$\sum_{t=1}^{T} O_t^C \leq K_C \tag{6.62}$$

6.5.6 The Model

The model for optimising the use of countermeasures over time consist of the objective function given in (6.12), and the set of constraints described in Sections 6.5.1 through 6.5.5. The Table 6.1 shows the constants, parameters, and variables used in the model.

Constants:	
H	Number of threats in the scenario
K_C	Maximum number of chaff dispensings
K_D	Maximum number of decoys to be deployed
M	A big number
T	Number of time steps in the given time frame
T_{JA}	The time it takes for the jammer to become active after being turned on
T_{JS}	The time it takes for the jammer to stop jamming after being turned off
T_{DA}	The time it takes for a deployed decoy to become active
T_{DR}	The time it takes to release a decoy
T_{CF}	The time it takes to form a chaff cloud after chaff has been dispensed
T_{CD}	The duration of a chaff cloud
T_{CL}	Latency time between two contiguous chaff dispensing
ε	Penalty value subtracted
Parameters:	
ρ_{th}	The distance between the aircraft and the threat h to the time t
ρ_{th}^*	The minimum of the distance and the value 2
α_{th}	The angle between the aircraft and the threat h to the time t
P_{th}	The probability of a radar h posing a threat to time t
R_{th}^J	The jammer reduction of lethality for threat h to the time t

Table 6.1: Constants, parameters, and variables used in the mathematical model.

R_{th}^{D}	The towed decoy reduction of lethality for threat h to the time t
R_{th}^{C}	The chaff reduction of lethality for threat h to the time t
Variables:	
O_t^J	Is 1 when the jammer is turned on
$O_t^{\prime J}$	Is 1 exactly when the jammer gets turned on
$O_t^{\prime\prime J}$	Is 1 exactly when the jammer gets turned off
C_t^J	Count for how long the jammer has been on
A_t^J	Is 1 if the jammer is active
O_t^D	Is 1 if a decoy is deployed
$O_t^{\prime D}$	Is 1 exactly when the decoy gets deployed
$O_t^{\prime\prime D}$	Is 1 exactly when the decoy gets severed
C_t^J	Count for how long the towed decoy has been deployed
A_t^D	Is 1 if the decoy is active
O_t^C	Is 1 when chaff are dispensed
A_t^C	Is 1 if a chaff cloud is formed
Z	The objective value
$S_{\text{sum,norm}}$	The survivability measure
$R_{th}^{\max}$	Maximum reduction of lethality for threat h to time t
L_{th}	The lethality of threat h experienced at time t
a_{th}^J	Determines if the jammer is mitigating threat h to time t
a_{th}^D	Determines if the towed decoy is mitigating threat h to time t
a_{th}^C	Determines if chaff is mitigating threat h to time t
a_{th}^N	Determines if the no countermeasures are mitigating
$\mathcal{P}$	The total penalty

Table 6.1: Constants, parameters, and variables used in the mathematical model.

The problem, which is hereafter referred to as the Countermeasure Optimisation Problem (CMOP), is described by the compiled model given below.

Maximize

$$Z = S_{\text{sum,norm}} - \mathcal{P}$$

Subject to

$$\mathcal{P} = \sum_{t=1}^{T} \varepsilon \cdot (O_t^J + O_t^{\prime J} + O_t^{\prime\prime J} + A_t^J + O_t^D + O_t^{\prime D} + O_t^{\prime\prime D} + A_t^D + O_t^C + A_t^C)$$

$$
\begin{aligned}
S_{\text{sum,norm}} &= \frac{1}{T}\sum_{t=1}^{T}(1-(\sum_{h=1}^{H} P_{th}\cdot L_{th}\cdot(1-R_{th}^{\max}))) \\
R_{th}^{\max} &\le R_{th}^{J}A_t^J + M(1-a_{th}^J) && \forall\, t,h \\
R_{th}^{\max} &\le R_{th}^{D}A_t^D + M(1-a_{th}^D) && \forall\, t,h \\
R_{th}^{\max} &\le R_{th}^{C}A_t^C + M(1-a_{th}^C) && \forall\, t,h \\
R_{th}^{\max} &\le M(1-a_{th}^N) && \forall\, t,h \\
\sum_{cm\in\mathcal{C}} a_{th}^{cm} &= 1 && \forall\, t,h \\
a_{th}^N &\le \rho_{th} && \forall\, t,h \\
a_{th}^N &\ge \rho_{th}^* - 1 && \forall\, t,h \\
O_t'^J &\ge A_{t+T_{JA}}^J - A_{t+T_{JA}-1}^J && \forall\, t \\
O_t''^J &\ge O_{t-1}^J - O_t^J && \forall\, t \\
\sum_{j=t}^{t+T_{JA}-1} O_j^J &\ge T_{JA}\cdot O_t'^J && \forall\, t \\
C_t^J &\le C_{t-1}^J + 1 && \forall\, t \\
C_t^J + T(1-O_t^J) &\ge C_{t-1}^J + 1 && \forall\, t \\
C_t^J - T\cdot O_t^J &\le 0 && \forall\, t \\
C_t^J &\ge 0 && \forall\, t \\
A_{t+T_{JS}}^J &\le 1 - O_t''^J && \forall\, t \\
T\cdot A_t^J &\ge C_t^J - T_{JA} && \forall\, t \\
\sum_{j=t}^{t+T_{JS}-1} A_j^J &\ge T_{JS}\cdot O_t''^J && \forall\, t \\
\sum_{j=t}^{t+T_{JS}-1} O_j^J &\le T_{JS}\cdot(1-O_t''^J) && \forall\, t \\
O_t'^D &\ge A_{t+T_{DA}}^D - A_{t+T_{DA}-1}^D && \forall\, t \\
O_t''^D &\ge A_{t-1}^D - A_t^D && \forall\, t \\
\sum_{j=t}^{t+T_{DA}-1} O_j^D &\ge T_{DA}\cdot O_t'^D && \forall\, t \\
O_t^D &\ge A_t^D && \forall\, t \\
C_t^D &\le C_{t-1}^D + 1 && \forall\, t \\
C_t^D + T(1-O_t^D) &\ge C_{t-1}^D + 1 && \forall\, t
\end{aligned}
$$

$$
\begin{aligned}
C_t^D - T \cdot O_t^D &\leq 0 && \forall\, t \\
C_t^D &\geq 0 && \forall\, t \\
T \cdot A_t^D &\geq C_t^D - T_{DA} && \forall\, t \\
\sum_{j=t}^{t+T_{DR}-1} O_j^D &\leq T_{DR} \cdot (1 - O_t''^D) && \forall\, t \\
\sum_{t=1}^{T} O_t'^D &\leq K_D \\
\sum_{j=t+T_{CF}}^{t+T_{CF}+T_{CD}-1} A_j^C &\geq T_{CD} \cdot O_t^C && \forall\, t \\
A_t^C &\leq \sum_{j=t-T_{CD}-T_{CF}+1}^{t-T_{CF}} O_j^C && \forall\, t \\
\sum_{j=t+1}^{t+T_{CL}} O_j^C &\leq T_{CL}(1 - O_t^C) && \forall\, t \\
\sum_{t=1}^{T} O_t^C &\leq K_C \\
t &\in [1, T] \\
h &\in [1, H] \\
O_t^J, O_t^D, O_t^C &\in \{0, 1\} \\
O_t'^J, O_t''^J, O_t'^D, O_t''^D, &\in \{0, 1\} \\
A_t^J, A_t^D, A_t^C &\in \{0, 1\} \\
C_t^J, C_t^D &\in \{0, 1\} \\
a_{th}^J, a_{th}^D, a_{th}^C, a_{th}^N &\in \{0, 1\} \\
L_{th}, S_{\text{sum,norm}}, R_{th}^{\max} &\in \mathbb{R} \\
Z, \mathcal{P} &\in \mathbb{R}
\end{aligned}
$$

6.6 The GAMS Program

The CMOP from Section 6.5.6 is implemented in a GAMS program. When run this program includes two files. The first file, `reductions.dat`, contains the reductions for the three countermeasures as functions of both angle (measured

in degrees) and range (measured in percentage of maximum range). The second file describes the flight for which the optimal survivability is to be found. It contains the distances and angles to the threats in the scenario for every time step in the timeframe. For each time step the lethalities for all threats are also described. The name of this file is given as an argument to GAMS when the program is run.

The GAMS program is included in Appendix D. Before the optimisation of the objective function is commenced the reductions of threat lethality for each countermeasure against every threat in each time step must be calculated. This is done by looking up the relevant values in the `reductions.dat` file included. Also the values of ρ^*_{th} are calculated for all threats at every time step. Finally the probability of every threat actually posing a threat is found for all time steps. The model is solved by GAMS using a Mixed-Integer Problem solver in CPLEX.

When a solution is found the results are added to a file. These results describe the date and time of the run, the number of time steps in the time frame, the objective value, the optimal survivability, the number of seconds used in finding the solution, and finally the name of the input file. These results are used in parts of the tests described in Section 6.7.

6.7 Testing

The testing of the GAMS program is conducted in three steps: first a number of test files are constructed to evaluate the time-related behaviour of the three countermeasures. Second a number of flights are generated from two scenarios. This is done to find the time it takes for GAMS/CPLEX to find the optimal solution depending on the number of time steps given in the flight. In the third and final step of the testing the optimal survivability to a number of different scenarios are found.

The scenarios described by the test files used in the first step of testing the mathematical model have very little resemblance with real world scenarios. To each time step described in these files fixed values for the distance and angle parameters are used. These values have been found to invoke responses from each of the countermeasures if no other constraints restrain them, and they are used to create a need for each countermeasure for a given period of time. Table 6.2 shows the pairs of parameter values used in the files, and the reductions given by each of the countermeasures using these values.

	$\alpha = 20°$ $\rho = 95\%$	$\alpha = 120°$ $\rho = 70\%$	$\alpha = 90°$ $\rho = 20\%$
R_{Jammer}	$0.93 \cdot 0.80 = 0.74$	$0.30 \cdot 0.77 = 0.23$	$0.28 \cdot 0.18 = 0.05$
R_{Decoy}	$0.43 \cdot 0.70 = 0.30$	$0.77 \cdot 0.83 = 0.64$	$0.70 \cdot 0.20 = 0.14$
R_{Chaff}	$0.05 \cdot 0.01 \approx 0.00$	$0.75 \cdot 0.10 = 0.08$	$0.80 \cdot 0.80 = 0.64$

Table 6.2: Reductions for each of the three countermeasures at fixed angles and distances.

To test the time-related behaviour of the countermeasures 14 test files are constructed. Four of these files (named `jamtest1.dat` to `jamtest4.dat`) test the behaviour of the jammer, the towed decoy is tested by five files (`dectest1.dat` to `dectest5.dat`), four are used to test chaff (`chftest1.dat` to `chftest4.dat`), and a single file (`mixtest1.dat`) describes three threats, each of which may be countered by one of the three countermeasures.

The need for a countermeasure is described by the lethality it may counter. For most of the files this lethality is set to 50% in the time steps where there is a need for the countermeasure, and 0% where the countermeasure is not needed. For the files `dectest5.dat` and `chftest4.dat` the lethality of the time steps where the countermeasure is needed is set to either 5%/50% or 3%/30%. This is done to test whether the countermeasure will be allocated for the time steps where it will offer the highest reduction of lethality. Table 6.3 shows descriptions of the 14 files and the results found running the GAMS program with the files as input. For every file an illustration of the lethality for each time step is given. This illustration also shows when the appropriate countermeasure is turned on and when it becomes active. It can be noted that if enough countermeasures are available they will be active when the lethality requires it.

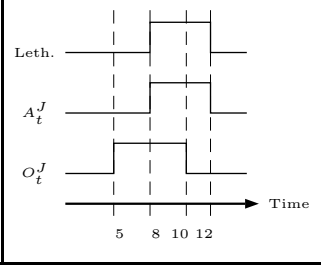
Description:	**Behaviour:**
File: `jamtest1.dat` The duration of the need for the jammer is long enough to test the constraints given by $T_{JA} = 3$ and $T_{JS} = 2$.	

Table 6.3: Descriptions of 14 files used for testing the mathematical model. *Continues...*

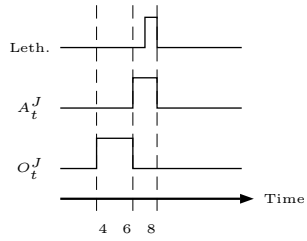
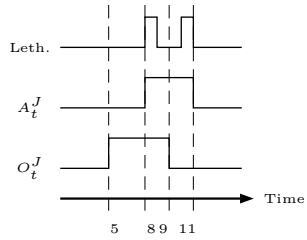
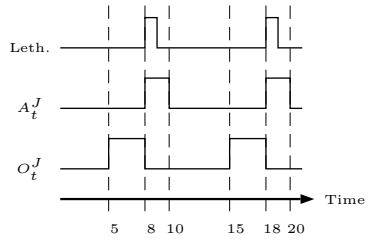
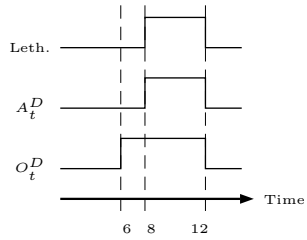
Description:	Behaviour:
File: `jamtest2.dat` The jammer is needed for less than $T_{JS} = 2$ time steps.	
File: `jamtest3.dat` The distance between two jammer requests is less than $T_{JA} + T_{JS} = 5$ time steps.	
File: `jamtest4.dat` The distance between two jammer requests is more than $T_{JA} + T_{JS} = 5$ time steps.	
File: `dectest1.dat` The duration of the need for the towed decoy is longer than $T_{DA} = 2$.	

Table 6.3: Descriptions of 14 files used for testing the mathematical model. *Continues...*

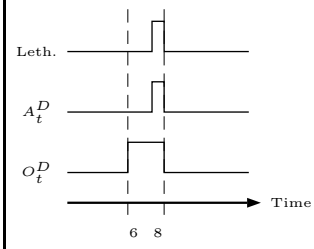
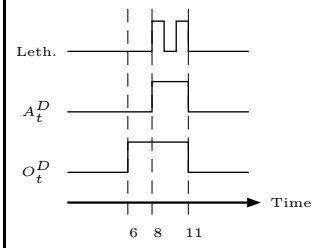
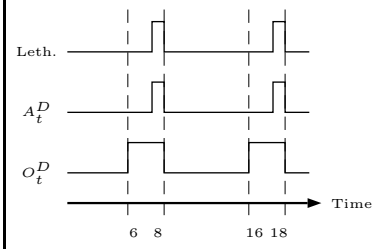
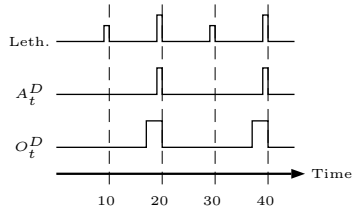
Description:	Behaviour:
File: `dectest2.dat` The towed decoy is needed for less than $T_{DA} = 2$ time steps.	
File: `dectest3.dat` The distance between two towed decoy requests is less than $T_{DA} + T_{DR} = 3$ time steps.	
File: `dectest4.dat` The distance between two towed decoy requests is more than $T_{DA} + T_{DR} = 3$ time steps.	
File: `dectest5.dat` More requests for towed decoys than decoys available ($K_D = 2$). Lethality varies.	

Table 6.3: Descriptions of 14 files used for testing the mathematical model. *Continues...*

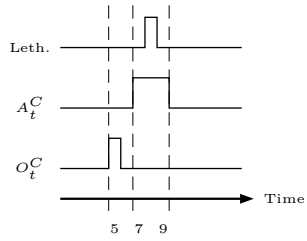
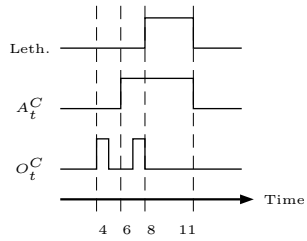
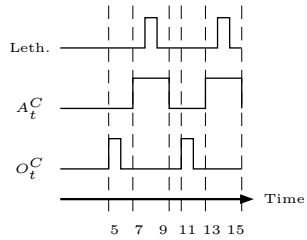
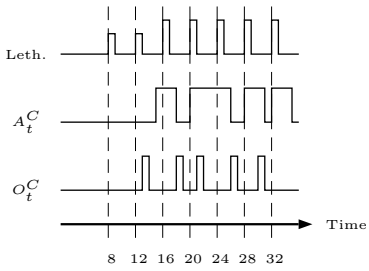
Description:	Behaviour:
File: `chftest1.dat` Chaff is needed for a single time step.	
File: `chftest2.dat` Chaff is needed for more than $T_{CD} = 3$ time steps.	
File: `chftest3.dat` More chaff dispensings requested.	
File: `chftest4.dat` Requests for more chaff than available ($K_C = 5$). Lethality varies.	

Table 6.3: Descriptions of 14 files used for testing the mathematical model. *Continues...*

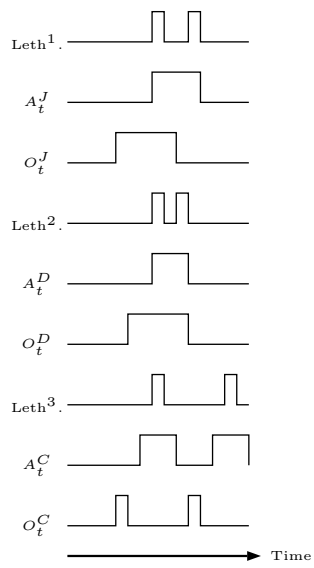
Description:	Behaviour:
File: `mixtest1.dat` A combination of the three files: `jamtest3.dat`, `dectest3.dat`, and `chftest3.dat`. Each file represents a threat.	

Table 6.3: Descriptions of 14 files used for testing the mathematical model. These files test the time-relations of the three countermeasures and they have very little resemblance to real-world scenarios.

Running the 14 files described in Table 6.3 shows that the time relations for the three countermeasures are satisfied with the constraints given in the mathematical model. For the jammer-related test files it is shown that the jammer is turned on T_{JA} time steps before becoming active, and it will remain active for at least T_{JS} time steps, or for as long as it is needed. If it can not be turned down and turned back on again between two requests it will remain turned on. If time allows, it will turn off between two jammer requests.

Similar results are found for the towed decoy. It is also shown that if there are more requests for a towed decoy than there are decoys available the decoys will be assigned to the request where they will offer the best reduction of lethality.

Testing the chaff relations show that a chaff cloud will be formed for as short a period of time as possible. As with the towed decoy the available amount of chaff will be assigned to the requests, where they yield the best reduction of lethality if more requests are present. Running the `mixtest1.dat` shows that

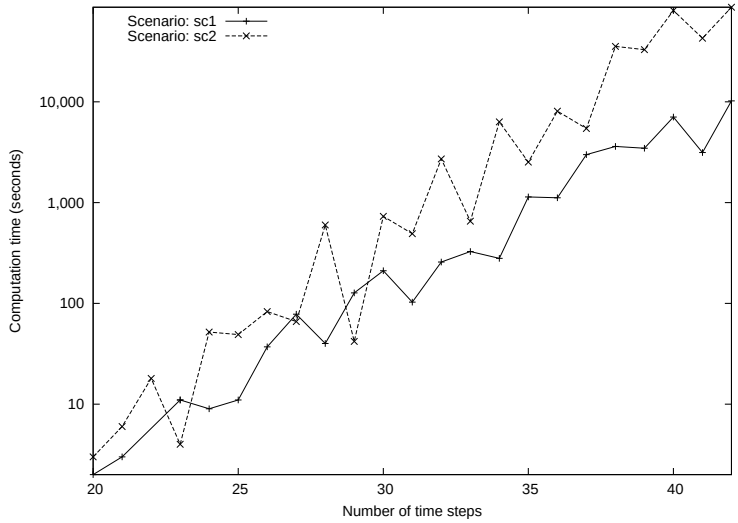
the three countermeasures are assigned to the three threats in the same way as if each threat was described in a file by itself.

The next part of the test is concerned with the relation between the number of time steps in a flight description and the time it takes to find an optimal solution to the deployment of countermeasures for that flight using GAMS/CPLEX. To do this a number of flights are generated from two of the scenarios, `sc1` and `sc2`, shown in Table 6.4. For each flight the distance and angle between the aircraft and a threat are sampled at a number of equidistant points in time. If it is assumed that the aircraft will maintain the same airspeed independent of the number of time steps in the description of a flight the time-related parameters, T_{JA}, T_{JS}, etc., must be adjusted according to the number of time steps in a flight and the distance the aircraft has to fly. To keep the generation of flights and the comparison of results easy it is chosen to keep the time-related parameters fixed for all runs. The results of this part of the test are illustrated in Figure 6.16.

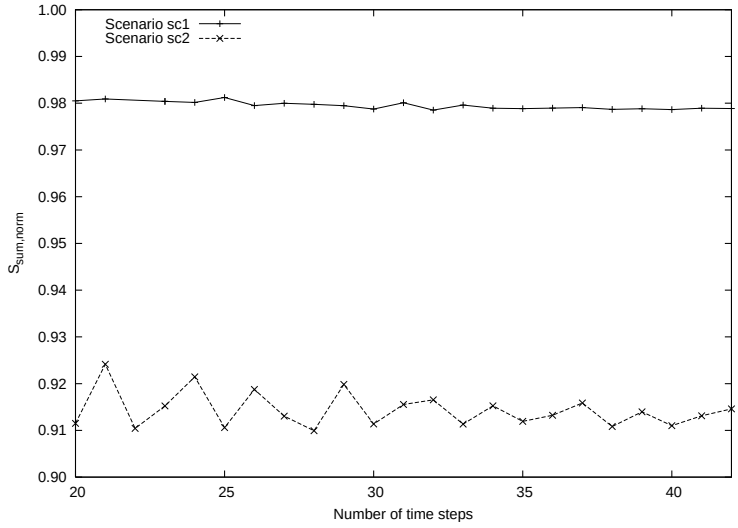
The graphs in Figure 6.16(a) show that the time it takes to find an optimal sequence for a flight increases with the number of time steps in the problem. The time axis has a logarithmic scale and even though the points in the graphs do not describe two perfect lines it is a reasonable conclusion that the running time for finding the best use of countermeasures is an exponential function of the number of time steps in the problem description.

The constraints in the model determine that the *NoCM* dummy countermeasure is chosen whenever the aircraft is not within the range of any threat. The main part of the problem solving remains when the aircraft is within range of one or more threats. To avoid boundary problems all the scenarios described in Table 6.4 are constructed so approximately the first third of a flight is flown outside the range of any threats. For the two scenarios used in constructing the graphs in Figure 6.16 the last third of the flight is also flown outside the range of threats. The time it takes to find an optimal solution for these two scenarios then largely depends on approximately one third of the number of time steps in the flight description. When sampling the flight with different numbers of time steps the number of time steps where the aircraft is outside the range of threats will vary. This is considered to be the explanation of the fluctuations within the time it takes to find an optimal solution.

Intuitively it may seem that the more time steps used in describing a flight the higher a survivability can be found. The graphs in Figure 6.16(b), showing survivability found as a function of the number of time steps in a flight, contradict this. When sampling a flight smaller threats or requests for certain countermeasures may fall in between sample points. Re-sampling the flight with more sample point may then reveal the before hidden threats and countermeasure



(a) Running time.



(b) Average survivability.

Figure 6.16: The running time as a function of the number of time steps in a scenario is shown in Figure 6.16(a). Note the logarithmic scale on the time axis. Figure 6.16(b) shows the value for $S_{\text{sum,norm}}$ as a function of the number of time steps in the flight.

requests, providing the pilot with better chances to counter the threats. Since new threats will introduce an increase in lethality the survivability found using the mathematical model is likely to fall when the sample rate is increased. This does not mean that the survivability for the pilot in the real world will decrease when knowledge of more threats is added to the problem description.

For the two scenarios, `sc1` and `sc2`, the increase in time steps does not reveal any new threats. For `sc1` the survivabilities seem to be almost constant regardless of the number of time steps. The survivabilities found for `sc2` has more fluctuations which can partly be explained by the changes in the number of time steps in which the aircraft is within the range of a threat. When more samples of the flight are taken these changes will become less critical and the survivability will tend to level out. This seems also to be the case for the survivabilities found for `sc2`.

In the last part of the test the optimal survivability is found for flights in six different scenarios. These scenarios, named `sc1` to `sc6`, are constructed to represent the following cases: dense sampling (`sc1`, `sc2`, `sc3`, and `sc4`), sparse sampling (`sc5` and `sc6`), more than K_D requests for towed decoy (`sc5` and `sc6`), more than K_C requests for chaff (`sc4`), and a "worst case" scenario (`sc6`). The scenarios are shown in Table 6.4.

The survivabilities found running the mathematical model using GAMS/CPLEX are listed in Table 6.5. When comparing the results in this table to the scenarios depicted in Table 6.4 it can be seen that increasing the number of time steps in where the aircraft is within range of one or more threats will decrease the average survivability for the flight. This is not surprising since the aircraft being within range of a threat will give an increase in the scenario lethality.

Another observation that comes as no surprise is that when the number of time steps where the aircraft is within range of a threat increases so does the penalty value found. The penalty value depends on the number of steps in which countermeasures are applied, and they will be so when the aircraft is within range of a threat.

The last column of Table 6.5 contains the most notable results. Here it can be seen that while finding an optimal solution to the `sc1` and `sc2` flights can be done within a few seconds, it will take more than six hours to find the optimal solution to the `sc3` flight, although the number of time steps in these three flights are identical. One of the differences between the scenarios is that while `sc1` and `sc2` each contains a single threat only the `sc3` scenario contains two threats. Finding results to the flights in `sc4` and `sc5` takes approximately an hour and a half, even though these scenarios contain three threats each. From this it can be seen that estimating the time it takes to find the optimal solution

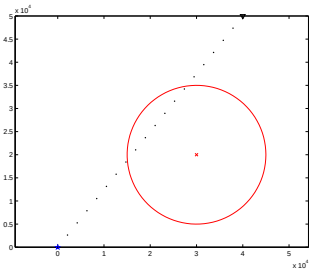
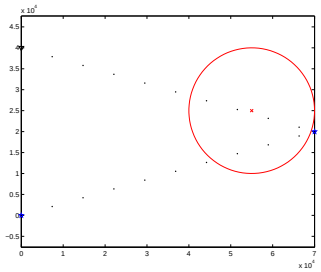
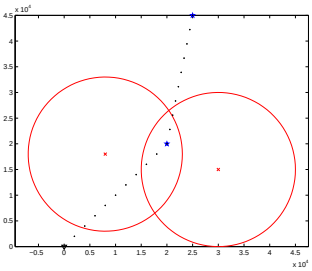
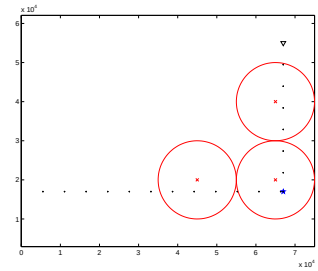
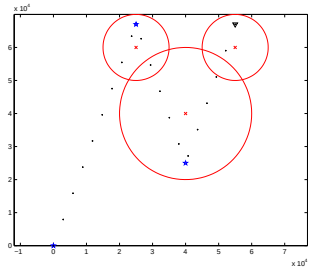
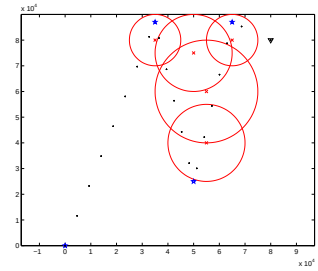
sc1 - Passing a single threat	sc2 - Turning at an IP
	
sc3 - Two threats	**sc4 - Three threats, many chaff**
	
sc5 - Three threats, many decoys	**sc6 - Many threats**
	

Table 6.4: The six scenarios used in the final part of the test. The flight in each scenario has 20 steps, and they all start outside the range of any threats to avoid boundary conditions.

Scenario:	$S_{\text{sum,norm}}$:	Penalty:	Running time:
sc1	0.98211	0.0016	2
sc2	0.91419	0.0027	6
sc3	0.91214	0.0037	22326
sc4	0.83773	0.0065	5858
sc5	0.83773	0.0065	5708
sc6	-	-	37916

Table 6.5: Results of solving flights for six different scenarios. Running time is measured in seconds. The process of finding a solution to the sc6 flight is terminated by the solver, and no solution is found.

to a flight is not trivial, as it apparently depends on more than the number of time steps in the flight, the number of time steps within range of a threat, and the number of threats in the flight scenario.

6.8 Discussion

In both the Prolog approach (Chapter 4) and the BN approach (Chapter 5) the focus was on finding the appropriate use of countermeasures at each point in time. With the approach described in this chapter it is assumed that the process of finding the best countermeasures to use at any given time can be reduced to a simple table lookup. In a real-world implementation of this approach the best countermeasure to apply at any time may be found by consulting e.g. a Prolog program or a BN.

If a countermeasure is to be active during the first few time steps of the time frame, solving the model will make the countermeasure being turned on before the start of the time frame. As this is not considered a correct answer, the flights are constructed in such a way that no threats are imminent in the beginning of the time frame. Doing this the model does not need to be concerned with boundary issues.

The aircraft can be protected by a countermeasure at any time. If neither towed decoys nor chaff are available an active jammer may always offer some reduction of lethality. This is only prevented if the jammer can not be turned on some time ahead. This happens only if the jammer is still active from a previous deployment interval. Extending this interval to fulfil the new requirements will solve the problem. Here boundary issues are neglected and it is assumed that the jammer can be turned on in any time step, even if this time step lies before the start of the time frame.

Compared to dispensing chaff the fiscal cost of deploying and releasing a towed decoy is very high. Even if the towed decoy offers only a slightly better protection than chaff, solving the model described in this chapter will select the towed decoy over chaff, not taking the cost of this into account. If the cost was incorporated in the objective function, such that chaff dispensing were preferred even if this results in a slightly worse reduction of lethality than using a towed decoy, the solutions found might be closer to what a fighter pilot will choose when in the same situation. Determining the allowable decrease in survivability to save the deployment of towed decoys is considered outside the scope of this work.

A large part of the work on the mathematical model is spend on determining the constraints describing the relations between the time steps in where a countermeasure is turned on and the time steps in where it is active. The survivability of the fighter aircraft depends only on when countermeasures are active, and not on when they are turned on. It may therefore be beneficial to leave the variables describing the latter out of the model. The period in which the jammer is turned on without being active sets a limit to how close jammer deployment intervals can be. It is assumed that this can be described using fewer and simpler constraints than the ones describing relations between the O_t^J and A_t^J variables. For the towed decoy the time between deployment intervals is set by the sum of the time it takes to settle after a decoy has been released, and the time it takes to unreel a new decoy. Here the need of a variable describing when a decoy is deployed is not evident either. Since chaff can be dispensed while the cloud from a previous dispensing is still formed, it may be that describing the time-relations for chaff can not be done by a single variable only. Still, omitting a large number of variables and constraints for the jammer and the towed decoy is likely to decrease the total computation time for finding an optimal solution.

According to Section 6.4.3 a total of 1500 time steps may be needed for describing a mission with a duration of up to five minutes, if the resolution of the description is set to five time steps per second. While this resolution may seem adequate, a coarser resolution can be acceptable. For the six scenarios used for testing the mathematical model, the number of time steps was chosen without considering the length of the flight and the time it will take to fly this. The flight in `sc1` is one of the shortest of the six flights. From start point to target this flight is approximately 64 km. With a speed of Mach 2 (2,124 km/h) a fighter aircraft will fly this distance in little more than 100 seconds. Decreasing the proposed resolution by a factor of 10, the time steps are set two seconds apart. With this the description of the `sc1` flight will require approximately 50 time steps; more than twice the number of time steps used in the tests described in this chapter. Descriptions of the other flights, with the `sc3` flight as an exception, will require even more time steps. Since GAMS/CPLEX in the default configuration can not be expected to solve larger problems within a reasonably

time, it may be necessary to use other configurations, or to split the problems into subproblems, that may be solved separately.

6.9 Conclusion

This chapter shows that for a simple model of a scenario containing ground based radar threats a survivability measure for fighter aircraft can be described. From this a mathematical model can be formulated to describe how a set of countermeasures influence the survivability. Both the survivability and the countermeasure influence are based upon the use of RF based threats only. Broader descriptions, e.g. including both IR and RF based threats, are likely to be significantly more complicated than the survivability measure and countermeasure influence described here.

A large part of the mathematical model describes the time relations of countermeasures. If other countermeasures are to be added to the model it is likely that some of the constraints constructed can be adaptable to work for these countermeasures as well. As described in Section 6.8 some of these constraints may not be necessary.

The use of GAMS/CPLEX described in this work will not allow for solutions to larger problems to be found within a reasonably time. Even for small and simple scenarios, as the six test scenarios described in this chapter, finding a solution takes a very long time. Using non-uniform time steps and weighting as described in Section 6.4.3 may relieve this issue. If the solving method can itself be optimized, so that problems in a real-world scenario can be solved in real-time, the success of this approach still relates to the availability of knowledge about the scenario. If the system has no knowledge about a threat prior to entering its lethal range, an optimal solution for the entire mission can not be guaranteed.

Even if solving the mathematical model can never meet the real-time requirement, the model may still have a number of useful applications. If a real-world scenario has a fixed number of threats reported by intelligence, a solution found pre-mission can be applied. A number of standard scenarios may also be defined, and the optimal sequence of applying countermeasures in these can be applied when needed. Chapter 7 describes the use of metaheuristics to find good solutions to the CMOP. Optimal solutions found using GAMS/CPLEX can be used in improving the performance of such a metaheuristic.

Chapter 7

The Metaheuristics Approach

A *heuristic* is an algorithm that uses some knowledge about a concrete problem in the process of finding a solution to it. Usually it can not be guaranteed that a heuristic finds the best solution to the problem, or even that the solution found is feasible. A *metaheuristic* is an algorithm framework that describes how to find solutions without being concerned with problem-specific considerations. These considerations must, naturally, be made when implementing a metaheuristic for solving a given type of problems. As a tailored metaheuristic is a heuristic the feasibility and optimality of the solutions found can not always be guaranteed. Metaheuristics are often applied for problems within the field of combinatorial optimisation, where they usually provide good, but not necessarily optimal, solutions. They have proven successful when no exact method for finding a solution is known, or when only a limited amount of computing time is available. Throughout this text the word "algorithm" refers to the metaheuristic it describes.

The choice of the metaheuristic to use in solving a problem, and setting up parameters for this, depends on several factors. These factors include the type of problem to solve, how close to optimum the solutions must be, and how fast the metaheuristic has to provide solutions. This chapter gives a general introduction to metaheuristics, some details on the *Simulated Annealing* metaheuristic, and comments on an implementation of this algorithm. Readers familiar with metaheuristics in general, and the Simulated Annealing specifically, may skip to

Section 7.4, where an implementation of the Simulated Annealing metaheuristic for solving the Countermeasure Optimisation Problem (CMOP) derived in Chapter 6 is described.

7.1 Motivation

With the CMOP the goal is to find the deployment scheme that provides the best survivability for a fighter aircraft during a mission. It is possible to calculate the total survivability for all possible instances of the deployment scheme, and then select the deployment scheme yielding the highest survivability. Unfortunately, this approach is infeasible for problems with more than a minimum number of time steps in the deployment scheme since the generation and evaluation of all the schemes will simply take too long.

In Chapter 6 solutions to the CMOP is found using mathematical programming. Using GAMS/CPLEX shows that an optimal solution can be found to some problems, and even though this can be done very fast compared to the time it takes to generate all possible deployment schemes, the time it takes to find an optimal solution is still too long. Therefore a solution is sought using a metaheuristic.

The fact that a metaheuristic can not be guaranteed to find an optimal solution is not considered important for finding a solution to the CMOP. The description of a battlefield scenario will often be so deficient that there is no way of deciding whether the exact optimal solution found is in reality superior to any other good solution. To the pilot any solution that significantly increases the chances of survival may be considered a good solution.

7.2 Metaheuristics

The *search space* of a problem is the set of all possible solutions to that problem. For a combinatorial optimisation problem the search space will consist of all combinations of values for the variables included in the problem. A problem suitable for a metaheuristic will usually have a search space that is too large to be searched completely. A metaheuristic will search parts of the search space in the pursuit of the optimal solution.

In general a metaheuristic first finds an initial solution, and in iterations better solutions are found, until it comes to a stop. In every iteration the current best

solution is known as the *incumbent*, and the solution being the incumbent at the end of the run is returned by the metaheuristic. In the search for improved solutions a number of *candidate* solutions are compared to the incumbent. New candidates are chosen from the *neighbourhood* of the incumbent. These candidates are also known as *neighbours*.

The metaheuristic runs for a number of iterations, and the incumbent may be replaced by a better solution in each of the iterations. The number of iterations is determined by one or more *stopping criteria*. The run may be stopped if the total number of iterations exceeds some preset limit, or if the incumbent has not been replaced for a fixed number of iterations. Also, reaching a maximum amount of computation time may be used as a stopping criterion.

In this work three metaheuristics are investigated: Local Search, Steepest Ascent, and Simulated Annealing. The three metaheuristics have been chosen for their simplicity, which makes them easy to implement. Steepest Ascent and Simulated Annealing can both be considered special cases of a general Local Search metaheuristic, and this makes the comparison of the three even easier, since the comparison can be done using the same program with only minor changes. Other metaheuristics such as Genetic Algorithms or Tabu Search may provide solutions of a quality equal to or better than those found using Local Search, Steepest Ascent, or Simulated Annealing. The intention of the work described in this chapter is to evaluate the use of metaheuristics for solving the CMOP. It is estimated that this can be done by implementing the three countermeasures mentioned, and therefore no other metaheuristics have been chosen.

7.2.1 Local Search

In the *Local Search* metaheuristic candidates from the neighbourhood of the incumbent are selected at random. If a candidate solution is proven better than the incumbent it becomes the new incumbent. The algorithm continues until no improving neighbour can be found, or until a maximum number of iterations have been run. The Local Search metaheuristic for maximising the objective function is given in Algorithm 1. Here $f(s)$ is the objective function that returns a value for the solution s.

This metaheuristic will search for the optimum local to the initial solution. If this optimum is reached, i.e. there is no neighbouring solutions that will improve the solution, the metaheuristic will stop. Applying the algorithm a number of times on the same problem may find a number of local optima from which the best optimum found can be selected.

Algorithm 1: The Local Search metaheuristic.

```
Find an initial solution s_0
inc ← s_0 ;                                  ▷ Initialise the incumbent
repeat
    s ← neighbour(inc) ;                     ▷ Random neighbour
    if f(s) > f(inc) then
        inc ← s
until stopping criterion is met
return inc;
```

If the search space of the algorithm consisted only of local optima, instead of solutions leading to a single local optimum, the optimum found may be closer to the global optimum, depending on the stopping criterion. This is the essence in *Iterated Local Search*, as described in [27]. Here a metaheuristic, e.g. Local Search, is applied to a starting point in the search space and a local optimum is found. The neighbourhood of this local optimum is then broadened and a new solution is found. From this solution a new local optimum is found. The difference between Iterative Local Search and just applying the Local Search numerous times with a random initial solution for each run is that the initial solutions for each new run of the Local Search algorithm within Iterative Local Search depends on the previously found local optimum.

The definition of the broader neighbourhood is essential to the success of the algorithm. If the relation between a local optimum and the starting point for the next iteration is too loose the effect would be the same as just applying the Local Search a number of times with randomly chosen initial solutions. If the relation is too rigid the starting point for the next iteration would likely lead to the local optimum the metaheuristic is trying to escape.

7.2.2 Steepest Ascent

The *Steepest Ascent* metaheuristic is very similar to the Local Search metaheuristic. The difference is that in Steepest Ascent all neighbours of the incumbent are evaluated, and the one offering the highest increase in the objective is chosen as the next incumbent. If no neighbour offers a better solution, a local optimum is found, and the algorithm stops. Algorithm 2 describes the Steepest Ascent, and $f(s)$ is again the objective function. A similar metaheuristic can be used in minimising the objective. This is known as the *Steepest Descent* metaheuristic.

Algorithm 2: The Steepest Ascent metaheuristic.

```
Find an initial solution s_0
inc ← s_0 ;                                   ▷ Initialise the incumbent
repeat
    best ← inc
    forall neighbours ∈ neighbourhood(inc) do
        s ← next_neighbour(inc) ;                          ▷ Next neighbour
        if f(s) > f(best) then
            best ← s
    inc ← best
until stopping criterion is met
return inc;
```

As with the Local Search the Steepest Ascent algorithm may be run multiple times from different starting points to at least find the best of a number of local optima, or it can be used as the metaheuristic in Iterative Local Search that will find local optima.

7.2.3 Simulated Annealing

The *Simulated Annealing* metaheuristic is described in e.g. [15, 35]. The algorithm is inspired by the physical annealing process of e.g. metals. In physical annealing the energy of a substance will decrease as the temperature of that substance falls. The lowest energy state for a metal is found when the atoms of the metal form a perfect crystal. However, when the metal is quenched, the atoms might settle in a non-optimal order, where the energy is not the lowest possible.

The laws of thermodynamics state that during the annealing of e.g. a metal, the probability of a an increase in energy, δE, at temperature T is given by $P(\delta E) = e^{-\delta E/kT}$, where k is Boltzmann's constant, $k = 1.38$ J/K. In the Simulated Annealing metaheuristic the neighbourhood of the incumbent is once again searched to find a solution better than the incumbent. If such a solution is found it replaces the incumbent, and the search continues. The analogy to the thermodynamical annealing is that a candidate solution that is not better than the incumbent may still replace it with a probability depending on how much worse this solution is. In this comparison a virtual temperature is also introduced. A starting temperature is given at the beginning of the metaheuristic run, and it decreases with the number of iterations. When the temperature decreases so does the probability of accepting a candidate solution worse than

the incumbent. The Simulated Annealing metaheuristic is shown in Algorithm 3, where $f(s)$ is again the objective function. This variant of the metaheuristic is first described by Metropolis, and it is explained in e.g. [15].

Algorithm 3: The Simulated Annealing metaheuristic.

```
Input: T_0 is the the start temperature
       γ is the temperature projection function
Set start temperature T ← T_0
Find an initial solution s_0
inc ← s_0 ;                                  ▷ Initialise the incumbent
repeat
    for i ← 0, niter do ;     ▷ Number of iterations per temperature
        s* ← neighbour(inc) ;                          ▷ Random neighbour
        if f(s) > f(inc) then
            inc ← s
        else
            δ ← f(inc) − f(s)
            Select x, x ∈ [0, 1]
            if x < exp(−δ/T) then ;      ▷ Accept if x is low enough
                inc ← s
    T ← γ(T) ;                                  ▷ Find next temperature
    Adjust niter
until stopping criterion is met
return inc;
```

The *niter* variable in the algorithm gives the maximum number of iterations at each temperature. This number can be adjusted during the run of the algorithm to find the best solution using the least computation time. More details on the use of this variable is given in Section 7.3.1.

In difference to the Local Search algorithm it is possible to escape a local optimum with Simulated Annealing, and it is thus possible to cover a larger area of the search space. In the beginning of a run the metaheuristic will allow for many solutions to become the incumbent, while it towards the end will almost only accept improving solutions.

7.2.4 Choosing a Metaheuristic

In implementing a metaheuristic most of the work is spend in defining constraints and variable relations, determining what constitutes the neighbourhood of a given solution, and how a neighbour is selected at random. Since these considerations are almost identical for the three metaheuristics described in this section, they alone do not suggest which metaheuristic to choose. In solving the CMOP the proper metaheuristic should provide a good solution within a very short time span. For every iteration of the Steepest Ascent metaheuristic it needs to generate and evaluate every neighbour solution. Since every solution may have a large neighbourhood, the Steepest Ascent algorithm may have performed a few iterations only, before it is stopped by a given maximum running time, and the best solution found may not differ substantially from the initial solution. The Local Search algorithm may perform a little faster, since it generates only the amount of neighbour solutions necessary for it to improve its best solution. The drawback of the Local Search is that it may spend most of its time on finding the optimum local to the initial solution, without any search for a better solution. The Simulated Annealing mends this, as it is designed to escape local optima at the beginning of a run. For large neighbourhoods it performs faster than the Steepest Ascent, and while it is deemed to be slower than the Local Search, it may escape local optima, and it thus have the possibility of finding better solutions.

Figure 7.1 shows the progress of improving the survivability for each of the metaheuristics. It can be seen that Local Search will reach a steady state after a relatively short time. The Steepest Ascent keeps improving its solution and no steady state is reached within the time given. In the first part of the time given, the survivability found by the Simulated Annealing fluctuates. It finds a steady state later than the Local Search, and the survivability found here is higher than that of either of the other metaheuristics.

According to [15] Simulated Annealing offers the best results on uniform data, i.e. data where very few clusters are present in the search space. This is in contradiction to a typical solution for the CMOP where each countermeasure will be active only within a number of deployment intervals, and these intervals often occur only when the aircraft is close to a threat. Both the deployment intervals and the presence of a threat can be considered as clusters in the solution space, and therefore Simulated Annealing may not show the best performance solving the CMOP.

In the literature on Simulated Annealing [15] the algorithm has been shown to give good results for most of the problems it has been applied to, albeit more specific heuristics usually perform better. If the problem at hand has too many

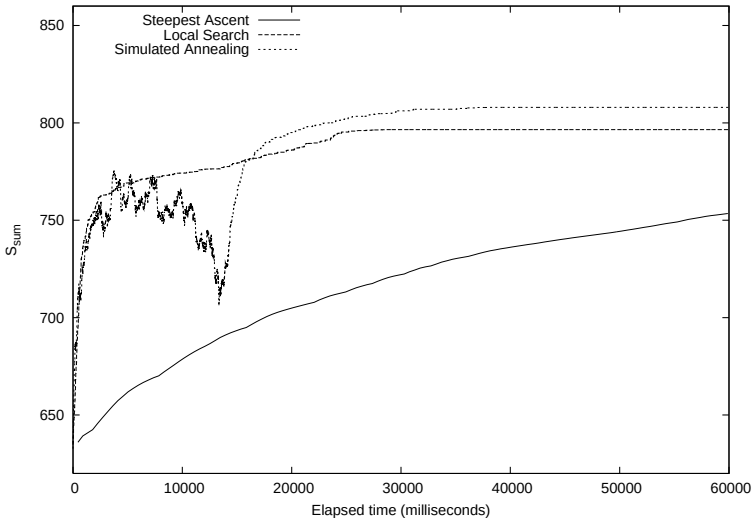


Figure 7.1: The survivability as function of time elapsed for Local Search, Steepest Ascent, and Simulated Annealing. The metaheuristics have been applied to a scenario `sc6` flight with 1000 time steps. A description of this scenario is found in Section 6.7 repeated in Section 7.5.

facets, e.g. many degrees of freedom, or many inter-variable relations, it may be difficult to design an efficient heuristic to solve the problem. In situations as this Simulated Annealing is relatively powerful. It has been shown that the algorithm has its best performance if given enough time. If it needs to find solutions fast, the best results are often found having only a single iteration at each temperature and a temperature decrease that is relatively small.

7.3 Using Simulated Annealing

In implementing a metaheuristic for solving a specific problem one has to decide on a number of details. These decisions can be split into two categories: the *generic decisions* which relate to the overall behaviour of the algorithm, and the *problem-specific decisions* that are concerned with finding solutions for the relevant type of problems [15]. Decisions within the two categories are discussed in this section.

In making decisions on the implementation of the algorithm, one may prioritize options that reduce the running time of the algorithm over options that improve

the objective value. The structure of the algorithm gives that converging toward a local optima is referred to the latter part of the algorithm run, and the success of the algorithm depends on the number of iterations it can perform. To perform a high number of iterations within a very short period of time requires that the parts of the code that are to be executed within every iteration must be written so it takes the shortest time possible to execute them.

7.3.1 Generic Decisions

The general decisions in implementing the Simulated Annealing are concerned with how the cooling is done, the parameters defining it, and how these affect the acceptance of solutions worse than the incumbent. These decisions are described here, together with criteria for stopping the metaheuristic running.

Algorithm Variant. In Algorithm 3 the metaheuristic performs a number of iterations at each temperature. In other variants a single iteration is done at each temperature. To get to the same amount of iterations the cooling is often slower when only a single iteration is performed at each temperature. While this cooling has the best resemblance to the physical annealing it is not evident which of these algorithm variants that offers the best results.

Cooling Schedule. In Simulated Annealing a cooling schedule describes the progress of the temperature decrease. In [15] a number of different cooling schedules are discussed, and here two of these are described. In the first schedule a geometric temperature reduction function, γ_G, is introduced. The temperature T_i in the i'th iteration is calculated from the temperature T_{i-1} at the previous iteration by $T_i = \gamma_G(T_{i-1}) = T_{i-1} \cdot a$. Here a is the *reduction factor*, and it will usually have a fixed value between 0.8 and 0.99. For a slower cooling the γ_S reduction function is suggested: $T_i = \gamma_S(T_{i-1}) = T_{i-1}/(1 + b \cdot T_{i-1})$. Here b is chosen sufficiently small to ensure a slow cooling. The graphs in Figure 7.2 shows the behaviour of γ_G and γ_S.

It should be noted that in adjusting the reduction factor a and the small value b the two cooling schedules can come to display very similar behaviour. If this behaviour satisfies the requirements one may have to solution quality within a fixed period of time, or within a given number of iterations, the schedule that requires the least number of arithmetical calculations can be selected to save computation time. Since γ_G involves a single multiplication it has a slight advantage compared to the combined addition and division of γ_S.

The history of using Simulated Annealing shows that in general it is less

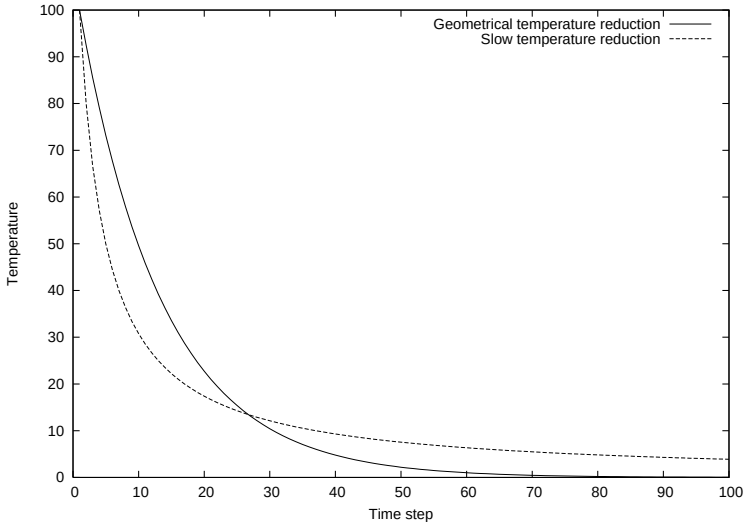


Figure 7.2: The behaviour of two different cooling schedules.

important how the cooling is done, and more important for how long the algorithm is allowed to search for improving solutions. The best parameter settings for the cooling schedule chosen are problem dependent, and they must be found in trials with representative problems [15].

The higher the reduction factor, the faster the metaheuristic will tend to dismiss solutions worse than the incumbent. This means that the metaheuristic will sooner focus on finding the local optimum. The reduction factor must be chosen in such a way that a sufficiently large amount of local optima can be explored.

The Metropolis formulation of the algorithm suggests that a number of iterations are performed at each temperature, before the temperature is reduced. The number of iterations can be fixed, or it may vary according to the temperature so that more neighbours are tested at lower temperatures where the probability of finding an improving solution is at its lowest. Varying the number of iterations can e.g. be done using geometrical or arithmetical projections, or by using a feed-back function. In finding the number geometrically the new number of iterations, i' is found by multiplying the old number of iterations, i, with a number b: $i' = i \cdot b, b > 1$. In the arithmetical approach the new number of iterations is found adding a fixed number to the old number of iterations: $i' = i + b$. If the number of iterations is to be found using feed-back, a limit can be set to the number of iterations where no improvement is done. Doing this the number of iterations at each temperature adapts to the solutions found. The num-

ber of idle iterations allowed may itself depend on the temperature; thus adding to the complexity of finding the best parameter settings for the algorithm.

Start Temperature. The behaviour of the algorithm during the start of the run depends highly on the start temperature given. The higher this temperature is the more solutions will get accepted during the start phase. If it is too high each new solution will be accepted, and the iterations of the metaheuristic will behave like a number of random solutions picked from the search space. If the start temperature is too low the ability to escape local optimum by accepting non-improving solutions will be decreased.

End Temperature. Choosing the end temperature is one way of selecting a stopping criterion. The faster the temperature reaches the end temperature the faster the algorithm is halted. On the other hand, if the end temperature is set too low, the algorithm may have many iterations at the end where there is no improvement in the solutions found.

Acceptance Probability. The difference between the incumbent and a candidate solution is used in determining if the candidate can be accepted. Letting the Boltzman probability, $P(\delta E) = e^{-\delta E/kT}$, decide whether a solution worse than the incumbent can be accepted as the new incumbent will keep the analogy to the physical annealing intact. While this probability may work in an implementation of Simulated Annealing, it may not be the best choice of acceptance probability.

Some implementations have shown that evaluating a new solution using the exponential function may use as much as one third of the total execution time [15]. Therefore it may be appropriate to find an acceptance probability not using the exponential function. In [15] two alternative acceptance probabilities are suggested. The first is $P(\delta E) = 1 - \delta E/T$. This probability approximates the exponential function with less computation time. The other suggested probability is found in a look-up table where the index idx is an integer value given by $idx = \text{round}(\delta E/T)$. These two acceptance probabilities, as well as the original Boltzmann probability, all display similar behaviour. Therefore the choice of acceptance probability can be reduced to finding the probability that may be calculated in the shortest period of time. Since the combination of rounding off a floating point number and using it as an index in a table is probably more time consuming than a single integer subtraction the first alternative acceptance probability suggested is preferred.

Stopping Criteria. The stopping criteria determine when the algorithm is stopped. A number of criteria can be defined, and the algorithm can come to a stop if either of these is met. If the algorithm is required to give

a solution within a fixed time the calculation time itself may be a stopping criteria.

The total number of iterations can also be used as a stopping criterion. In general the more iterations performed by a metaheursistic, the closer the returned solution will be to the optimum. On the other hand, having the metaheurstic running for a very long time may allow only few changes in the objective towards the end, thus having only minor improvements in the survivability. The number of iterations in which the solution has not improved may also be used as a stopping criterion.

7.3.2 Problem-specific Decisions

Three guidelines to making problem-specific decisions are mentioned in [15]: First of all, the algorithm should remain valid. Second, computation time must be used effectively, so as many iterations as possible can be executed, and improvements can be found throughout the execution. Finally the solution returned by the algorithm must be close to the optimal solution. This can be ensured by tuning parameters and comparing the solutions found to optimal solutions found using an exact mathematical solver. Some problem-specific decisions are described here.

The Objective Function. The objective function should preferably be easy and fast to calculate, since it is to be done in every iteration in comparing the incumbent with a candidate solution. If calculating the objective function is very time consuming, it can be beneficial to introduce a function that approximates the objective function, if this approximation is faster to calculate. To ensure that the algorithm is converging towards the optimum of the objective function, and not only the optimum of the approximation, the value of the objective function can be used for evaluating both the incumbent and the candidates at a given interval.

For some problems it is possible to calculate the change in the objective value by the differences between the incumbent and the candidate solution without having to calculate the complete objective function for the candidate. For S_{sum} and $S_{\text{sum,norm}}$, as defined in Section 6.4.2, the updated objective function can be found by calculating the contribution from the time steps that is different between the incumbent and the candidate, subtract this amount from the objective value, and then add the contribution from the candidate.

If it is hard to find feasible solutions, some of the constraints that make most solutions infeasible can be relaxed. This means that new solutions

can be generated without complying with these constraints, and a penalty for violating the constraints are subtracted from the objective function. Since solutions that violate the excluded constraints may not give as good results as the ones not violating the constraints, due to the subtracted penalties, the probability of these being accepted will drop towards the end of the run. The use of penalty values was also introduced in Section 6.5.1 to regulate the behaviour of some of the involved variables. The objective function to choose is then composed of two parts. If the objective is to maximise the objective function, the first part represents a relevant value which is to be maximised, and the second part is the penalty that will be minimised, or hopefully disappear, when the algorithm approach the global optimum.

Neighbourhood. The success of the algorithm highly depends on the definition of the neighbourhood, and how a neighbour solution is chosen. In [15] it is mentioned that changing the definitions of the neighbourhood during the run may also improve the results of the algorithm. In the beginning of a run the probability of accepting a worse solution is high. In much the same way the neighbourhood may be defined broader in the beginning, to allow for a larger part of the search space to be investigated. In the last part of the run, where the algorithm focus on finding the optimum local to the incumbent, the neighbourhood needs to be more narrow. Only solutions in the vicinity of the local optimum will then be evaluated.

A solution might have more neighbour solutions than possible to evaluate within a reasonable amount of time, and therefore a neighbour solution is often chosen at random. Since the functions to find a solution at random are often very time consuming, [15] suggests that neighbours are taken in order to avoid the use of randomness.

Initial Solution. The choice of initial solution may influence the solutions found by the algorithm. With an appropriate initial solution the metaheuristic may converge towards a good solution very fast. On the other hand, a good initial solution may trap the metaheuristic in a local optimum that can be difficult to escape.

If the parameter settings are chosen so that almost any candidate solution is accepted in the beginning of a run, the choice of initial solution is less important. With a good initial solution the parameters can be set so that fewer candidates are accepted, and the algorithm can find a local optimum fast. Doing this will make the Simulated Annealing behave like an ordinary Local Search algorithm.

7.3.3 Parameter Tuning

Both the generic decisions and the problem-specific decisions require settings for a number of parameters to be found. Finding the best parameter settings is itself a combinatorial problem, since every parameter may have many different values possible, and the set of parameters found for one problem may not be the best set to use on another problem; especially if the two problems are very different. Therefore the problems need to be divided into classes, e.g. based on the size of the state space, and the set of parameters must then be tuned for each of these classes.

Parameter tuning is best done on a selected set of problems representing their class. If optimal solutions are known to these problems the parameters must be tuned so that the algorithm will return values as close to these as possible. Since solutions found by the metaheuristic will not be better than the known optimal solutions the process of parameter tuning can be stopped when the solutions found by the algorithm are considered close enough to the optimal solutions.

The best solutions can be expected if the Simulated Annealing is allowed to run for a very long time. This may not always be possible, and often the algorithm can be stopped earlier with no significant decrease in the quality of the solution returned. If the algorithm is allowed to run for a limited period of time only it must be able to improve the solution as much as possible within the limited time. To ensure this, parameters describing other stopping criteria, such as the end temperature or the maximum number of iterations, must be set so that the algorithm is not stopped unless a good solution is found.

In [15] it is suggested that the start temperature is found using a *heat up* process. This means that the algorithm is run a number of times with increasing start temperatures. When the increase in start temperature does no longer improve the solutions found by the algorithm, the start temperature to use has been found. In a similar way the end temperature can be found by a *cool down* process.

Parameters such as the start temperature and the reduction factor may be tested in combination. A high start temperature and a low reduction factor, making the temperature reduce fast, may show to have the same effect as a relatively low start temperature and a high reduction factor. The best combination of these parameters may be found by running the algorithm with a selection of combinations. This selection may be based on experiments in where an interval of interesting parameter values is found. If these experiments show e.g. that the reduction factor must have a value in the interval from 0.85 to 0.90 it will not be necessary to test for combinations where the reduction factor is set outside

this interval.

In [15] it is encouraged to display a graphical representation of the intermediate solutions from the algorithm during or after the run. Seeing which obstacles that stop the algorithm from improving solutions may help the metaheuristic programmer in deciding on the parameter settings. For the CMOP a text based representation can show the deployment scheme of the incumbent. If this scheme is too big to fit on a computer screen, it can be minimised so that each time step is represented by a single pixel only, the colour of which indicate the status of the countermeasures. If this minimised version is overlaid a map showing the threats in the scenario the status of each countermeasure can easily be related to the position of the aircraft relative to the threats.

7.4 Implementing Simulated Annealing

This section describes an implementation of the Simulated Annealing algorithm to solve instances of the CMOP described in Chapter 6.

7.4.1 The Objective Function

Some constraints may be relaxed to ease the process of finding feasible solutions. For the CMOP the constraints to relax can be the maximum number of decoy deployments or chaff dispensings in a solution. The penalty for violating these constraints may depend on the degree of violation, so having four deployment intervals for the towed decoy, when only three are allowed, has one penalty, while having five intervals will result in a more severe penalty, etc. The penalty for each interval above the limit must be larger than the maximum increase in survivability for an interval, so an infeasible solution will not be preferred to a feasible solution.

With the survivability function chosen it is possible to calculate the change in the objective function between the incumbent and a neighbour solution by inspecting the differences between the solutions only. Toggling the status of one of the countermeasures in the deployment scheme will result in changes in a limited amount of time steps surrounding the time step in where the toggling takes place. The number of involved time steps depends on the definition of the neighbourhood and on the time-related parameters for the selected countermeasure. Finding the new objective value is done in these steps:

1. Toggle the status of the countermeasure at the selected time step. Do the necessary synchronizations related to this.
2. Determine the interval of time steps involved in the synchronization.
3. Calculate the contribution of these time steps to the objective value of the incumbent, and the contribution from the same steps in the neighbour solution.
4. Subtract the contribution from the time steps in the incumbent, and add the contribution from the candidate.

If the aim is to only find the change in the objective value, e.g. to see if the candidate improves the solution found, the last step may be omitted.

7.4.2 An Initial Solution

For any scenario it is possible to find a feasible solution, since not applying any countermeasures at any given time will always be feasible, although the survivability from this solution may be far from the optimal value. Another initial solution can be found by deploying random countermeasures at random time steps throughout the time frame. For this solution to be feasible the number of deployment intervals for the towed decoy and chaff should be limited to the numbers available. This can be done by eliminating a number of intervals at random, if the number exceeds the expendables available. A more suitable variant of this initial solution can be found by limiting the time steps for assigning countermeasures to the time steps where the aircraft flies within range of one or more threats.

An even better initial solution can be found by first finding the countermeasure yielding the best reduction of the lethality for each threat at any time step, and then make this solution feasible. The following steps will result in a feasible initial solution:

1. Find the best countermeasure to every time step t during the mission. If for a given time step the aircraft is out of range of each threat, and none of the countermeasures thus provide an increase in survivability, none is selected.
2. Introduce timing constraints. If a countermeasure is needed to time t, it should be turned on ahead of t, and it should be turned off afterwards. Make sure that countermeasures are active for a minimum of time steps.

3. The feasibility of the solution found in the first two steps must be ensured. Infeasibility occurs if either the towed decoy or chaff is deployed more times than possible. If e.g. the towed decoy can be deployed three times only, and the solution found in the first two steps requires four deployments, two neighbouring deployment intervals are merged, or one of the intervals are removed. The two intervals to be merged can be chosen as e.g. the first two instances in the deployment scheme, or they can be chosen as the two deployments with the smallest distance in between. The same technique can not be used to reduce the number of chaff dispensings, since each chaff cloud formed will last for a fixed period only. Here the relevant number of dispensings are simply removed from the deployment scheme, until a feasible amount is reached.

7.4.3 Neighbourhood

In implementing the Simulated Annealing algorithm the choice of neighbourhood is essential to the results found. The definition of neighbourhood must not prevent any feasible solutions from being investigated, and with the acceptance of solutions worse than the incumbent any feasible solution must be reachable from any other feasible solution. This will ensure that the optimal solution can be found regardless of the initial solution chosen.

A neighbour solution can be found in numerous ways. One way is to pick a countermeasure and a single time step at random, and then invert the state of the incumbent for that countermeasure in that time step. To investigate larger parts of the search space one may select a larger number of contiguous time steps for each neighbour. When the algorithm approaches the end temperature it is likely that a neighbour very different from the incumbent will have an objective value much worse than that of the incumbent. Since this neighbour is unlikely to be accepted a neighbourhood only containing neighbours close to the incumbent is preferred in the end of the run. Therefore the number of contiguous time steps defining a neighbour can be decreased as the temperature falls.

When the countermeasure is chosen at random the probability of choosing one can be different from the probability of choosing another. These probabilities may depend on e.g. the current threat scenario, the amount of expendable countermeasures available, or the cost of deploying a given countermeasure. In a similar way the time steps can be selected so that e.g. time steps that will split a towed decoy deployment interval will have a lower probability of being selected, since they will result in more deployment intervals, i.e. increase the number of towed decoys used.

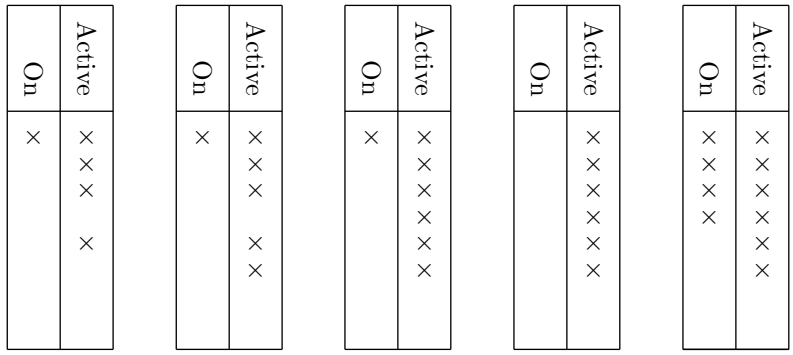
Searching a solution for an appropriate interval of time steps may prolong the time it takes to find a neighbour. This has to be considered when choosing the neighbourhood definition. If the algorithm is to run for a limited time only a slow neighbour selection will decrease the number of iterations that can be done. On the other hand, carefully choosing a neighbour may increase the probability of the neighbour being accepted. For this implementation of the Simulated Annealing algorithm the countermeasures will have equal probabilities of being picked. For every neighbour found a single time step is chosen, and the probabilities for choosing each of the time steps are equal.

The states of each countermeasure are described by two variables, one indicating if the countermeasure is *on*/*deployed*/*dispensed* and one indicating if it is *active*. In choosing a neighbour one has to decide which of these variables to invert when countermeasure and time step have been found. When a variable has been inverted the solution needs to be *synchronized* to become feasible. In the synchronization a number of *on* and *active* is set to make the solution feasible. It is assumed that the synchronization is done so that variables for a minimum number of time steps are affected. Synchronizing a solution after e.g. the jammer is set to be *on* at a given time step will have the jammer being *on* until it becomes *active*, and it will remain so for a minimum number of time steps. For the towed decoy there is no minimum time span in which it must remain *active* after being turned *on*. This means that if the decoy is turned *on* it will be *on* for a fixed length period of time, only to have the *on* status stopped before ever becoming *active*. For this reason it is chosen that only the *active* variable for each countermeasure can be inverted when finding a neighbour.

In relation to the synchronization an *island* is a deployment interval that is too short for the solution to be feasible. Since the jammer has to be active for at least T_{JS} time steps a deployment interval shorter than this is considered an island for the jammer. If the distance between two contiguous deployment intervals is too short for the solution to be feasible this interval is called a *gap*. When a towed decoy is released at least $T_{DR}+T_{DA}$ time steps must pass before a new towed decoy can become active. This can not be achieved if the time elapsed between two deployment interval is too short.

Synchronizing the solution when the *active* status of a countermeasure is inverted depends on the countermeasure chosen. For the jammer this is done in four steps as illustrated in Figure 7.3. For the towed decoy and chaff similar steps are performed during synchronization. The four steps for synchronizing the jammer are:

Extend introduced islands and gaps. If the period in which the jammer is set *active* is shorter than T_{JS} it may form an island. For the island to be



(a) *Active* in single step. (b) Period extended. (c) Gap removed. (d) *On* status removed. (e) *On* status replaced.

Figure 7.3: The steps required to synchronize the jammer part of the deployment scheme when the status of the jammer is set to on.

formed no time step surrounding the period must have the status set to *active*. If an island is found the period is extended until a minimum size of T_{JS} time step is reached. Similar actions are taken if the *active* status is removed during the period. Here the *active* status must be removed for at least T_{JA} time step.

Close gaps and remove islands. When a period of *active* is introduced gaps may occur before or after the period. If gaps are found they will be closed by setting the status to *active*. In a similar way islands may appear when the *active* status is removed. These islands are removed by removing the *active* status for each time step in the islands.

Remove *on* status Since changing the *active* status of a number of time steps will change when the jammer has to be turned on the *on* status is first removed from all affected time steps. This is done whether the synchronization is done after setting the jammer *active* or after removing the *active* status.

Set *on* status For all affected time steps the *active* status is checked. If the jammer is set *active* the *on* status in previous time steps are set appropriately.

7.4.4 Parameter Settings

The implementation of the Simulated Annealing algorithm offers a number of parameters to be set. The parameter settings will influence the performance of the algorithm and the solutions found. The parameters include the start temperature, the temperature reduction factor, generation of a non-empty initial solution (see Section 7.4.2), and the description of four different stopping criteria. The algorithm will stop when a maximum number of iterations is reached, if the current temperature gets lower than a preset end temperature, if the incumbent has remained unchanged for a number of iterations (referred to as *idle iterations*), or if a maximum running time has elapsed.

In determining the parameter settings two of the six scenarios used for testing the mathematical model (see Section 6.7) are used. The `sc1` scenario contains a single threat only, and it is chosen for determining the parameter settings as it is likely that finding solutions to flights in this scenario is relatively fast. This is explained by the fact that the time it takes for the implementation of the algorithm to calculate the objective value in each iteration depends on the number of threats given in the scenario, and the `sc1` scenario contains a single threat only. The second scenario for determining the parameters is the `sc5` scenario. This is chosen since it contains three threats, and it is thus likely that finding solutions to flights in this scenario will take an increased amount of time compared to finding solutions to flights in `sc1`. Since a valid optimal solution to the flight in scenario `sc6` is not found solving the mathematical model using GAMS/CPLEX, this scenario is not used in tuning the parameters for the algorithm. Short descriptions of the six scenarios are repeated in Table 7.2.

It is essential that the algorithm provides good results within a very short period of time. Therefore the algorithm is first run with varying values for the maximum running time. The time limit is set to vary from 10 milliseconds to 120 milliseconds. To ensure that most of the runs are terminated by the maximum running time stopping criterion, parameters describing the other three stopping criteria are set to refrain these from stopping the run. To perform the tests the start temperature is set to 1,000,000, the maximum number of idle iterations is 100,000, the end temperature is fixed at 10^{-20}, and the reduction factor is set to 0.995. The survivabilities found for varying time limits are shown in Figure 7.4.

For flights in scenario `sc1` runs with maximum running times above 65 milliseconds are stopped as the end temperature is reached before the maximum running time has elapsed. None of the flights show an increase in survivability when the maximum running time is set higher than 85 milliseconds. To leave a

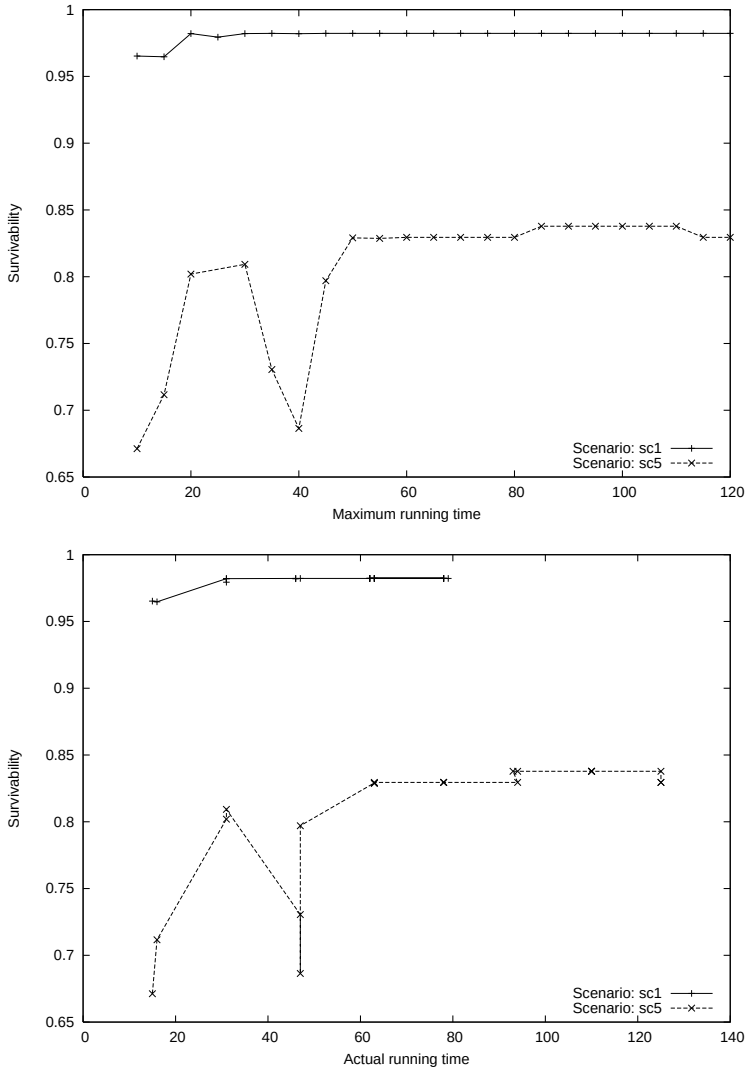


Figure 7.4: The survivabilities found for flights in scenarios `sc1` and `sc5`. The graphs show the survivabilities found as functions of the maximum running time and the actual running time, respectively.

margin for finding solutions to more complex scenarios the maximum running time is set to 100 milliseconds. This value is well within the 200 millisecond limit given in Section 3.3.

For a given start temperature the reduction factor must be set so that a good solution is found before the maximum running time has been reached. If the reduction factor is set too low, i.e. the temperature decreases too fast, the algorithm will not get to search larger parts of the search space before converging towards a local maximum. For the `sc1` and `sc5` flights the search spaces are relatively small, and large parts of them can be evaluated within relatively few iterations. Here the problem of having a small reduction factor is that it will make the algorithm reach the end temperature before the maximum running time is reached. This limits the number of iterations in which the solution can be found. Since it is assumed that having the algorithm run for as many iterations as possible will in general increase the objective value it is preferred that the algorithm is not stopped by other stopping criteria than the maximum running time. Setting the reduction factor too high will keep the temperature high throughout the execution of the algorithm. At high temperatures almost any candidate solutions will be accepted to replace the incumbent, and no good final solution can be guaranteed.

Combinations of start temperatures and reduction factors are tested on the `sc1` and `sc5` flights. These tests show that varying the parameter values have very little effect on the survivabilities found. For the `sc5` flight a reduction factor of more than 0.997 gives a decrease in survivability, while reduction factors below 0.993 will make the algorithm stop as it reaches end temperature. There are no apparent differences in neither the survivabilities found nor the stopping criterion evoked when the temperature is varied between 100 and 5,000,000. For the `sc1` flights having start temperatures below 500,000 result in the maximum time being reached only when the reduction factor is above 0.997. Increasing the start temperature to 5,000,000 does not change this. From these results the start temperature is chosen at 500,000. The reduction factor must be set to a value between 0.993, where the solving of the most time consuming problem is stopped at the end temperature, and 0.997, where the quality of the solutions is decreasing. The reduction factor is set at 0.995, even though finding a solution to the `sc1` flight is not stopped by the maximum running time criterion at this value. It is assumed that values above 0.995 is likely to decrease the quality of the solutions found, and it is assessed that better solutions are preferred even though they are found without invoking the maximum running time stopping criteria.

In determining the parameter settings an empty initial solution is being used. Section 7.4.2 describes another initial solution where the countermeasures offering the best reduction of lethality are set active at each time step. Solutions

Parameter:	Setting:
Start temperature	500,000
Reduction factor	0.995
Non-empty initial solution	No
Maximum number of iterations	100,000
End temperature	10^{-20}
Maximum idle iterations	10,000
Maximum running time	100 milliseconds

Table 7.1: The parameter settings used in testing the results found by the implementation of the Simulated Annealing metaheuristic.

to the `sc1` and `sc5` flights are found both with and without the non-empty initial solution. It is found that there are no significant differences between the solutions found when the initial solution is empty and when a non-empty initial solution is introduced.

The parameter settings are listed in Table 7.1.

7.5 Testing

The implementation of the Simulated Annealing algorithm has been exhaustively tested to ensure that it provides solutions that fulfil the constraints described in Section 6.5. The tests described in this section compare the solutions returned by the implementation of the metaheuristic to the optimal solutions found using GAMS/CPLEX as described in Section 6.7. The tests are performed on flights from the six scenarios described in Table 7.2

For each of the scenarios a 20 step flight is generated. Besides the 20 step flight for the `sc6` scenario a 1000 step flight is also generated. The parameter settings given in Table 7.1 are applied in finding solutions to each of the flights generated. For the 1000 step flight a different set of parameter settings is also applied. For each flight and set of parameter settings the implemented metaheuristic is run ten times. The results of these runs are given in Table 7.3.

Running the metaheuristic on flights from `sc1` to `sc4` finds only survivabilities similar to those found by solving the mathematical model. The ten runs for the `sc5` flight return two different values, and therefore the average survivability found here is lower than the optimal survivability found from solving the mathematical model.

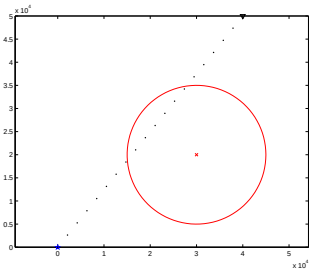
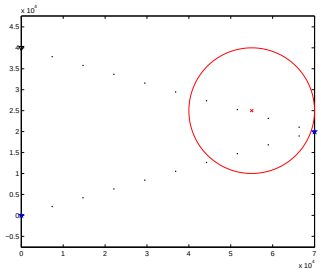
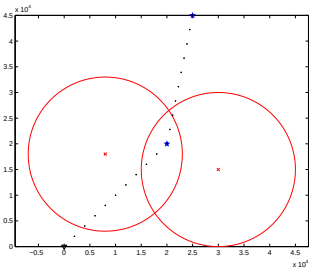
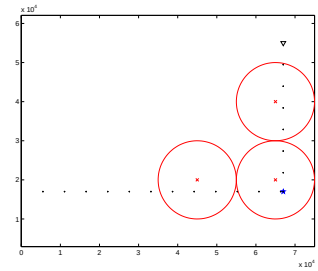
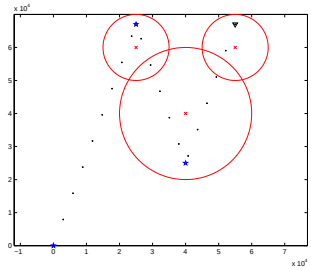
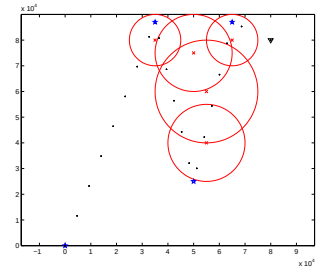
sc1 - Passing a single threat	sc2 - Turning at an IP
	
sc3 - Two threats	**sc4 - Three threats, many chaff**
	
sc5 - Three threats, many decoys	**sc6 - Many threats**
	

Table 7.2: The six scenarios used in the final part of the test. The flight in each scenario has 20 steps, and they all start outside the range of any threats to avoid boundary conditions.

Scenario / time steps:	Optimal surv.:	Average surv.:	Best surv.:	Idle iterations (average):
sc1/20	0.982	0.982	0.982	8403.4
sc2/20	0.914	0.914	0.914	8369.0
sc3/20	0.912	0.912	0.912	5140.8
sc4/20	0.837	0.837	0.837	644.3
sc5/20	0.837	0.833	0.837	887.6
sc6/20	-	0.813	0.817	1.9
sc6/1000	-	0.675	0.679	1.7
sc6/1000*	-	0.792	0.804	77.2

Table 7.3: Results of running the implemented Simulated Annealing metaheuristic on flights from the six scenarios described in Table 7.2. The values in the second column are found solving the mathematical model described in Section 6.5. The *-marked problem has been run with an alternative set of parameter settings.

The sc6 scenario has more threats than any of the other scenarios. Since the time it takes to perform a single iteration depends on the number of threats in the scenario, fewer iterations can be performed for sc6 flights, before the algorithm is stopped by the maximum running time criterion. This can be seen in the difference between the average survivabilities and the best survivabilities for the sc6 flights. These runs have very few idle iterations before the algorithm is stopped, indicating that a good solution can not be guaranteed.

The sc6 flight with 1000 steps is solved using both the same parameter settings as for solving the other flights, and with an alternative set of parameter settings. In the different parameter settings the maximum number of idle iterations is set to 1,000,000, the start temperature is set to 50,000,000, and the algorithm is allowed to run for 10,000 milliseconds. While these settings are in no way tuned to the solving of problems of this size, the results show that allowing the algorithm for more iterations will increase the survivability found. This illustrates the claim stated in Section 7.3.3 that the set of parameters must be tuned for each class of problems.

7.6 Discussion

The three metaheuristics described in Section 7.2 are all rather simple to implement. For finding a good solution to the CMOP it is not necessary to use more complicated algorithms, since the uncertainties on the input are substantial, and hence the difference between the solution in a local optimum and the global

optimum may be "lost" in uncertainties. Using simulated annealing offers an acceptable trade-off between solution quality and computing time.

Each iteration will include an evaluation of a new candidate. When reaching low temperatures very few of these candidates will be accepted. To save computation time in this part of the algorithm, the probability of accepting a new candidate can be estimated, and then be used in accepting or rejecting candidate solutions, without performing the time consuming evaluation of each of these. Estimating the probability depends on both the function for calculating the probability and on the current candidate. This is not a trivial task to do.

It has been proven that the Simulated Annealing algorithm will converge towards a global optimum given the right circumstances [40]. First, the Metropolis variant of the algorithm is to be used, i.e. a number of iterations are performed at each temperature. If the temperature is decreased cautiously, and the number of iterations at each temperature is large enough, the solutions found at each temperature will reach a *thermal equilibrium*. This means that while the optimal solution may not have been reached, the sum of signed fluctuations in the objective values will be close to zero. If thermal equilibrium is obtained at each temperature, and the Simulated Annealing is given unlimited time, the probability of achieving one of the optimal solutions is one. For this to be true the neighbourhood must be chosen in such a way that any feasible solution is reachable from any other feasible solution.

Although it is theoretically possible for the Simulated Annealing algorithm to converge towards a global optimum, it can not be assumed to do so solving the CMOP. One of the reasons for this is the real-time requirement that any system solving the CMOP must fulfil. The 200 milliseconds time limit for finding a threat response (see Section 3.3) does not match the unlimited time required for the Simulated Annealing to converge towards a global optimum.

While it is possible to reach all feasible solutions from all other feasible solutions it may not always be easy to do so. If for instance the incumbent contains a number of deployment intervals for the towed decoy equal to the number of decoys available, another deployment interval can only be introduced if one of the existing intervals gets eliminated. Removing a decoy deployment interval requires it to be removed step by step from either end. Toggling the state of the decoy at a time step within the interval will split the interval into two intervals. If the number of intervals is already at the maximum number allowed, this solution is not feasible. Having a neighbourhood where candidate solutions can eliminate an entire deployment interval completely will make it easier for the metaheuristic to escape local optima.

In the mathematical model, described in Chapter 6, the penalty for having a

countermeasure turned on or active in a given time step is $\varepsilon = 0.0001$. The small value is used because the optimal value, disregarding the penalty, would be relatively close to the value returned by GAMS/CPLEX. For the metaheuristic this small value may not be an advantage. In the first part of the run a solution with countermeasures turned on without being necessary, e.g. when the aircraft is outside the range of any threat, will only present a slightly worse solution than the incumbent, and the probability of it being accepted is relatively large. This may result in the introduction of unnecessary deployment intervals, and as explained above these may not easily be removed.

As with the mathematical model a large part of the effort in implementing the metaheuristics has been focussed on synchronizing the time steps where a countermeasure is *active* with the time steps where it is *on*. While this synchronization seems to work, it has not been tested for all possible combinations, and situations may exist in which the synchronization will return a non-feasible solution. Leaving out the variables describing when the jammer is turned on and when the towed decoy is deployed, as suggested in Section 6.8, will make the synchronization both easier to implement and faster to run.

The Simulated Annealing algorithm can be used in combination with other algorithms and metaheuristics. An example of this is the algorithm described in Section 7.4.2 for finding an initial solution. The Steepest Ascent algorithm can be used in a post-processing stage of the results given by the Simulated Annealing. If the Steepest Ascent algorithm is run long enough it is ensured that at least a local optimum is found.

Since no countermeasures should be deployed outside the range of threats in the scenario, omitting most of the time steps where the aircraft is outside all ranges from the problem description would reduce the size of the problem, thus improving the efficiency of the algorithm. Removing all these time steps can effect the solution found since a countermeasure may need to be turned on/deployed/dispensed ahead of the period in which it is within the range of a threat. Due to the limited amount of towed decoys aboard the aircraft it may also be necessary to keep a decoy deployed when the aircraft flies between the ranges of two threats. Care should therefore be taken if it is decided to reduce the problem size by omitting part of these steps.

During flight the solutions found at a given time step can be shifted one time step and be used as initial solution for a run in the following time steps. Doing this may lessen the time it takes to find good solutions for the following time steps and hence the allowed running time can be decreased. When shifting the solution one time step at each run, already found deployment intervals can be shifted out of the deployment scheme. When this happens to a decoy deployment or a chaff dispensing the total number of allowed deployments/dispensings must

be increased accordingly.

The tests described in Section 7.5 show how the number of threats influences the number of iterations that can be performed within a limited amount of time. This influence is caused by the calculation of the objective function, since this is the only part of the algorithm where the number of threats has an impact. If finding the objective value was done more efficiently, e.g. by following the steps described in 7.4.1, the effect of multiple threats will decrease.

7.7 Conclusion

With the implementation of the Simulated Annealing it is shown that it is possible to find good solutions to the CMOP within short time. To flights with no more than 20 time steps the solutions found will often be either optimal, or close to an optimum. For problems involving more time steps or threats than the ones tested in Section 7.5 the current implementation of the Simulated Annealing algorithm may still be too slow. No actions have been taken to improve the solutions for these problems within the time limit.

For the flights constructed good solutions are found within 100 milliseconds. It is believed that good solutions may also be obtained for larger problems if the parameter settings were optimised according to the size of these problems.

It is concluded that Simulated Annealing is an appropriate choice of metaheuristic for solving the CMOP. It is, however, not evident that it is the metaheuristic best suited to perform the task. To find the metaheuristic that will find the best solutions within a limited time will require more metaheuristics to be implemented, and more tests to be performed.

Chapter 8

Comparing Approaches

The four approaches described in this work represent four different technologies that can be used in designing a DSS for fighter pilots. In this chapter the four approaches are compared, and pros and cons of each technology are described. The approaches are compared with regards to five of the six design requirements described in Section 3.3: real-time, hardware, updateable, trustworthy, and usable. For each of these requirements the approaches are marked from one to five where five is the best. These marks are the result of a subjective assessment by the author, and from the marks the approach best suited for further development is found. The sixth design requirement, a usable user interface, has been omitted in this comparison, since no effort has been done to develop a suitable interface to either of the systems developed.

8.1 The Approaches

In Section 3.3 it is estimated that a DSS must be able to suggest evasive actions to the pilot within 200 milliseconds from the time a change in the scenario occurs. It is assumed that this response time is obtained running the DSS on an aircraft computer. Since neither of the systems developed in this work has been run on such a computer all the response times reported are estimates based on

the computation times found using the computer systems described in Appendix E.

For all approaches it is assumed that uploading mission data and updates to the systems can be easily done during the pre-flight preparation of the aircraft. How this is done is beyond the scope of the work.

8.1.1 Prolog

The Prolog program described in Chapter 4 uses a set of rules for a pilot to follow to perform evasive actions in a hostile environment. These rules are combined with a knowledge base containing descriptions of threats and how they are mitigated. Descriptions of the current scenario are also used by the program for suggesting actions.

Real-time. Traditionally the execution of Prolog programs is considered slow. This is due to the way a Prolog interpreter will compare every state obtained to a number of rules to see whether a rule can be matched. The program described in Chapter 4 has been run using B-Prolog on the laptop PC described in Appendix E. With this configuration neither of the runs was registered to take more than 150 ms. To this period of time the time it takes to register data about the current scenario must be added, as must the time it takes to present the output of the program to the pilot. In this chain of actions running the Prolog program is considered to be the most time consuming. It should be noted that adding to the complexity of the scenario or to the size of the knowledge base will increase the time used by the Prolog interpreter. This may influence the ability for the Prolog program to give real-time performance and hence it is rated the mark 4.

Hardware. Although it is unlikely that a Prolog interpreter is currently installed on an aircraft computer it is assumed that writing such an interpreter is relatively easy (see Section 4.1). Input to the Prolog system will come from either onboard systems, such as the MWS and the RWR, or it will be given pre-mission. The developed system requires data to be written to Prolog files which may be consulted at a fixed interval. Converting data from devices attached to the aircraft data bus to either a Prolog file or directly input to the Prolog interpreter is assumed a minor task. All in all the Prolog approach receives the mark 4 for fulfilling the requirements set by the aircraft computer and hardware.

Updateable. The Prolog program was developed in a relatively short time. This suggests that both minor updates and major rewrites can also be

done in a short time. If it can be assumed that the logic behind the Prolog program does not change, the updates necessary will be limited to the knowledge base and the programs for dispensing expendables. In the Prolog program these data are kept in separate files and the updates concern these files only. The Prolog approach is rated the mark 5 for fulfilling the updating requirement.

Trustworthy. The Prolog based DSS will suggest every feasible solution to all problems presented to it. Whether these problems are the results of real-world threats, or merely stems from false alarms from e.g. the MWS, is of no concern to the program. Input can come from different sources, each of which has a probability of supplying erroneous information and false alarms. If input from one source is erroneous the output from the program is likely to be erroneous too. The probability of the program suggestion actions that does not match the real-world scenario is thus bigger than it is for each of its input sources to deliver erroneous input. This problem has been described in [52].

The experienced pilot will have a perception of how reliable each of the on-board sensor system is, and he will respond to the warnings given by such a system according to this reliability. With the introduction of the Prolog program the sources of warnings will be hidden from the pilot, and the perception of reliability will be gone. For fulfilling the requirement of a DSS to be trustworthy the Prolog program is rated the mark 3.

Useful. The program will find all feasible actions to scenarios given. There is no ordering of the solutions, and the task of finding the best applicable solution is left to the pilot. The more threats the system needs to find actions for the more actions it will probably find. Since more threats in the scenario will increase the pilot's workload, an increase in the information presented to him by the system is not beneficial.

The Prolog system will register if a previously deployed countermeasure may currently have an effect on threats in the scenario. Also the amount of expendables is registered for the system to see if a given countermeasure program can be executed. Knowledge about threats that the aircraft will encounter in the near future is not used by the program. Thus neither the best sequence of deploying countermeasures, nor the probability of having enough expendables for the rest of the mission is found.

The Prolog program is basically categorical, and although any relations can be described with some uncertainty, it does not take the uncertainty on the observations into account. Unlike the other three approaches the Prolog program may improve the probability of the survival of the aircraft without giving an estimate of the improvement. If the notion of survivability was introduced in the program it may help in ordering the solutions

found at each time step. The overall usefulness of the Prolog program is rated the mark 3.

8.1.2 Bayesian Network

For the EW domain the variables in a BN refer to e.g. the presence of radar radiation, approaching missiles, or warnings issued by sensor systems on-board the aircraft. For each relation between a variable and the variables it depends directly upon the dependencies are described using a dependency table. When chances to the probability of a variable being in a given state is changed the BN is updated, and all related dependencies are re-calculated. For the BN the survivability is the probability of the variable Survive being in the *Survive* state. The best action yields the highest survivability.

Real-time. The time it takes to update a BN depends on the number of variables and dependencies in the BN. Therefore expanding the BN to be able to accommodate more scenarios and a broader range of actions is likely to slow down the execution. Speeding up the execution may be done by writing a program that is dedicated to finding the best combination of actions with the model developed, and not be dependent on the HUGIN tool.

For finding the best combination of actions to a given scenario each combination is set using the HUGIN user interface and the survivability from this combination is then calculated. Each of these calculations is done in what appears to be real-time. The number of combinations to evaluate is fairly small and it is assumed that the total time it will take to calculate the survivabilities for all combinations is well below one second, and probably within the 200 ms limit. This can be verified by writing a program that interacts with HUGIN and performs all the survivability calculations; this has not been done. Overall the BN approach is rated the mark 3 for the ability to fulfil the real-time requirement.

Hardware. The HUGIN software may not easily be ported to an aircraft computer. If HUGIN, or another PC-based BN tool, is used for the development of a BN, the model itself may be ported to an aircraft computer. Writing a program for this computer than can read the ported model and maintain and update a BN may be just as easy as the implementation of a Prolog interpreter as previously mentioned. The mark 4 is given to the BN approach for fulfilling the hardware requirement.

Updateable. Developing and maintaining a BN has proven to be very cumbersome. The dependency tables involved may become very large, and

updating these by hand is difficult. If statistical data about the relations between variables in the BN are available it may be possible to update the BN, or parts hereof, using Structural Learning.

Introducing new threats or countermeasures to the BN is done by introducing new variables or new states in existing variables. This can not be done without updating large parts of the BN since all variables being directly influenced by the changes must also be changed. All in all the BN approach gets the mark 2 for the ability to be updated.

Trustworthy. One of the advantages with the BN developed is that it is designed to manage the probabilities of sensors reporting false alarms. Even if the probability of a sensor issuing a warning is 100%, the probability of this being wrong is incorporated in the model. As long as the model used for the BN can be considered trustworthy so can the results found. For this the BN approach is given the mark 5. One may argue that the BN developed is not entirely trustworthy, and that the mark should be no more than 3 or 4. It is deemed that the model is as trustworthy as the domain knowledge upon which it is build, and therefore the mark 5 is obtainable if sufficient data are given.

Useful. For every scenario presented to the BN examining combinations of possible actions will result in the combination of actions yielding the highest survivability. This offers an improved usability compared to the Prolog program since the workload of the pilot is not increased by the DSS if the number of threats in the scenario increases.

Since expanding the BN is cumbersome, the number of e.g. missiles that is known by the BN is very limited. This means that the probability of the pilot encountering threats not known by the BN is relatively high. While this will not prevent the system from finding a good combination of actions, the pilot may not be convinced that this combination is in fact the best. The number of actions in the BN is also very limited, and this too will influence the usefulness of the BN.

When a manoeuvre is necessary for the aircraft to obtain a break lock the system will not describe this manoeuvre. Adding this functionality to the system would clearly improve its usefulness. Without it the usefulness of the BN-based DSS receives the mark 3.

8.1.3 Mathematical Model

The mathematical model is implemented to solve a problem different from the problem solved using the Prolog program or the BN model. Where the focus in the first two approaches is on providing solutions to the current threat scenario

the focus with the implementation of the mathematical model is on finding the best sequence of countermeasure deployments during an entire mission. Finding the best solution to any given time is considered trivial and is basically done by simple table look-ups. It is assumed that some knowledge about threats that will be encountered by the aircraft in the near future is known. The mathematical model is written in the GAMS language and it is solved using the CPLEX solver.

Real-time. Using GAMS/CPLEX solving problems with approximately 20 time steps will take from a few seconds to several hours, depending on the complexity of the scenario. This suggests that solving the mathematical model to optimality with currently available hardware and software can not be guaranteed in real-time. For this the approach gets the mark 1.

Hardware. Using the newest computer technology for running the GAMS/CPLEX software the computation times experienced in this work may be reduced substantially. It is unlikely that this technology will reduce the computation time enough for the system to become real-time, and it is equally unlikely that the technology will soon be implemented in existing fighter aircraft. For meeting the requirement that a DSS must be able to run on an aircraft computer this approach gets the mark 1.

Updateable. Changing the lethality of the involved threats, or adding new types of threats to the scenario, is relatively easy, as long as the new threats behave similar to the existing threats. Also changing the reduction functions for each countermeasure can be done with a minimum of effort since these functions are described using simple look-up tables.

Adding more countermeasures to the mathematical model will result in the addition of constraints to the model. As long as the added countermeasures have time relations comparable to those of the already included countermeasures this is a trivial task. This kind of countermeasures can either be described using the five-phase description from Section 6.5.1, or if they are expendables the constraints can be compared to those related to chaff dispensing. Another very important requirement to new countermeasures is that they work on RF based threats. If that is not the case the lethality will need to be redefined. The related reductions of lethality will also need to be computed as functions of the electromagnetic band in which they work.

Since updating the mathematical model with new threats and threat scenarios is likely to be the most frequent updates to the system the approach using the mathematical model gets the mark 4 for the ability to be updated.

Trustworthy. If it is assumed that knowledge about the type and positions of enemy threats are the results from intelligence reports, the solutions found

using the mathematical model will not suffer from the dependency of onboard sensors as e.g. the Prolog program does. As long as the description of threats used to find the optimal sequence of actions matches the real world the results can be trusted. As this can not always be guaranteed and information from e.g. the RWR is needed to update the threat scenario the system becomes less trustworthy and the mark is set to 3.

Useful. Since the performance of a system based on finding the optimal solution to the mathematical model is far from real-time the usefulness of such a system appears as quite small. It is assumed that positions of threats are known before the aircraft embarks on the mission. If this information is available long enough for GAMS/CPLEX to provide a solution before the mission is initiated this solution may still be useful to the pilot. The optimal solutions found solving the mathematical model can also be used in tuning the parameters for use in the metaheuristic. The mathematical model thus proves its usefulness in other areas as well.

In Section 2.3 it is mentioned that IR guided missiles may be considered the greatest threat towards aircraft. Since the mathematical model does not find solutions to scenarios including IR guided missiles the usefulness of the mathematical model diminishes.

The results from the system will describe when a countermeasure needs to get deployed, and which threat it is mitigating. How this deployment is to be done, and which manoeuvres that must accompany it is not given. This brings the usability of the mathematical model down to the mark 2.

8.1.4 Metaheuristics

The implemented Simulated Anneling metaheuristic will solve the same problems as solved by the mathematical model. The differences between these methods are mainly found in the value of the solutions, and the time it takes to find them. Where GAMS/CPLEX will use a long time to find the optimal value to a problem, the metaheuristic will often find a suboptimal solution to the same problem in much less time. As mentioned in Section 7.1 the description of a battlefield scenario will often be so deficient that the optimal solution found by the GAMS/CPLEX method may not be any better than any solution found using the Simulated Annealing metaheuristic.

Real-time. One of the advantages of a metaheuristic such as Simulated Annealing is that the time it is allowed to find a good solution can be used as a stopping criterion, and when stopped the currently best found solution is returned. This means that Simulated Annealing can probably be

used to find good feasible solutions within the 200 ms limit set in Section 3.3. Since the solution to the problem at a given time can build upon the solution found at the previous time step, the time it takes to find a good feasible solution will be even less than the times found in Chapter 7. For this the mark for the real-time requirement is set to 4.

Hardware. The success of the Simulated Annealing depends on the number of iterations it is allowed to perform before being stopped. For a fixed running-time this number of iterations depends on both the program implementing the metaheuristic and the hardware on which it is run. Even if an aircraft computer does not perform as fast as the laptop PC used for running the implementation described in Chapter 7 it will not render the use of the metaheuristic impossible; only the quality of the solutions found may be deteriorated. The mark for matching the hardware on-board the aircraft is set to 3.

Updateable. Updating the scenario for the metaheuristic can be done by updating the files that describes the scenario and the reduction of lethality by the countermeasures. The format of these files is the same for the implementation of the metaheuristic as for the files that are included in the GAMS program. Updating the program itself to include other countermeasure or to be able to counter IR guided missiles will require parts of the program running the metaheuristic to be rewritten. Since the updating of the implementation of the metaheuristic is similar to updating the system using the mathematical model it too gets the mark 4 for the ability to be updated.

Trustworthy. Compared to the system using the mathematical model the metaheuristic system can, due to its ability to deliver real-time solutions, respond to changes in the scenario as they occur. This will bring the mark for trustworthiness up to 4.

Useful. This system suffers from the same drawbacks as the system solving the mathematical model. That it may operate in real-time increases the usability of the system and it receives the mark 3 for this.

8.2 Comparison

The marks given for each of the five design requirements to the four approaches are collected in Table 8.1. Comparing the approaches can also be done by studying the web plots in Figure 8.1. Generally the best approach will be the one with the largest area in the web plot.

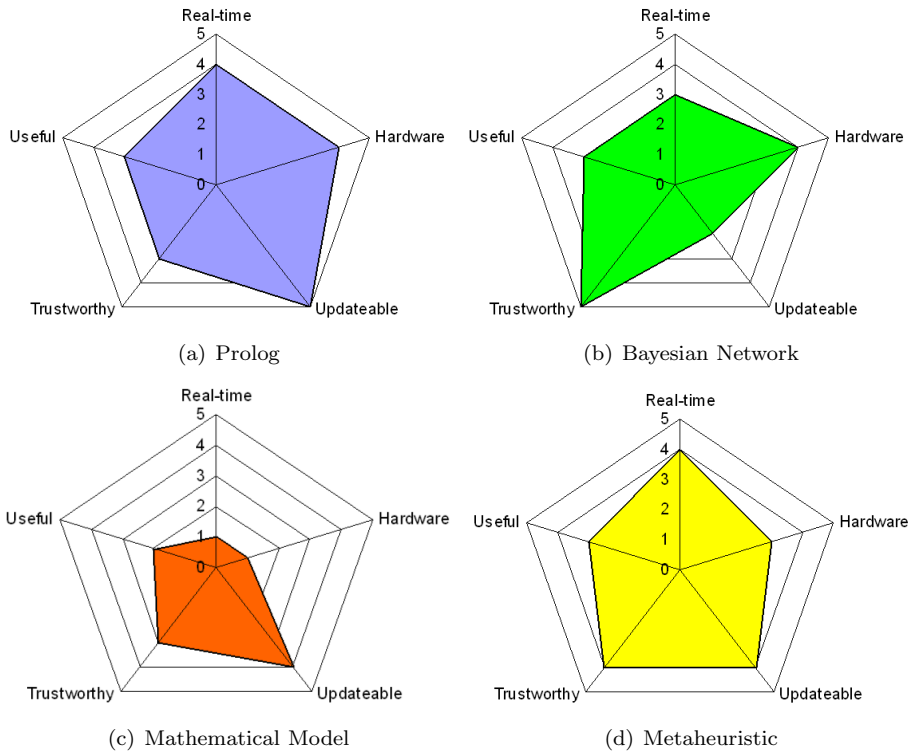
Requirement:	Prolog:	Bayesian Network:	Math. Model:	Meta-heuristic:
Real-time	4	3	1	4
Hardware	4	4	1	3
Updateable	5	2	4	4
Trustworthy	3	5	3	4
Useful	3	3	2	3

Table 8.1: The marks given for each approach with regards to the design requirements formulated in Section 3.3.

It is alluring to choose the best approach for building a DSS by simply selecting the approach with the best average mark for the five requirements. Using a weighted average may give a better impression of the strength of the different approaches. Finding this weighting is not trivial: to the DSS programmer it may be important that the system can offer real-time performance while working on an aircraft computer; the crew responsible for preparing the aircraft before a mission may find the ability to update the system to be important, while the pilot may prefer the system to be both trustworthy and useful. For the web plots in Figure 8.1 weighting the requirements differently is equivalent to scaling the axes for the requirements thus adjusting the area of the web plot.

Finding the best approach for a continuance of this work may depend on a requirement not previously mentioned: potential. Which of the four approaches has the highest potential of being a success with further development? Further development with the Prolog program may improve the usability of the program. This may also expand the number of predicates necessary to explore when finding a solution, thus possibly leading to increased computation time. For the BN approach the success depends on the possibility of updating dependency tables. If data for a Structural Learning (SL) procedure can be provided, this approach is likely to succeed. Although computers and algorithms for finding solutions to mathematical programs keep improving it is unlikely that the GAMS/CPLEX approach in near future will be able to perform real-time for problems of a relevant size. Since results from GAMS/CPLEX can be used to improve the performance of the metaheuristic the potential of the approach lies in increasing the number of time steps in the problems solvable. With the metaheuristic implementation the major disadvantages are that it will not find solutions to scenarios involving IR based threats, and finding the appropriate countermeasure at every time step is done by a simple table look-up. If these disadvantages can be alleviated the potential of the approach is improved.

It is the opinion of the author that a combination of the four approaches may



(a) Prolog

(b) Bayesian Network

(c) Mathematical Model

(d) Metaheuristic

Figure 8.1: Web plots illustrating the distribution of the marks given to the four approaches.

give the system best suited for further development. The time requirements renders the mathematical model per se useless as the technique applied in a DSS. The combination of solving the mathematical model to optimality and using the results to tune parameters for the metaheuristic seems viable, and the metaheuristic will be the best choice of technique for allocating countermeasures over time. Both the Prolog program and the BN amend the disadvantages with the metaheuristic implemented: they find solutions to scenarios involving IR based threats, and they find a solution to each time step by scrutinizing the descriptions of the scenario. If a sufficient amount of data can be established to build the BN using SL this is the preferred technique for finding the best combination of actions to each time step. This is due to the fact that actions suggested by the BN is based on the probability of input data being wrong. If data for building the BN can not be obtained a Prolog program will find the best actions to each time step. As stated earlier a Prolog program may be used to construct data for building the BN.

Chapter 9

Further Work

This chapter suggests further work for improving the results from the four approaches previously described. Also methods for testing the results from the approaches are described, and finally a few approaches which may be good candidates for further investigation are suggested.

9.1 Current Approaches

The work with the four approaches has been focussed on providing enough material to evaluate the concept of each approach as the technology to use in a DSS for fighter pilots. To fully understand the potential of these approaches further work needs to be done for each of them. Besides improving the understanding the tasks described will either improve the usefulness of each approach or lessen the time it takes to suggest actions to the pilot. Some of these tasks are described here.

9.1.1 Prolog

In Chapter 4 it is mentioned that defining the set of rules to build the Prolog program upon is best done in iterations, where an implementation of the current set of rules is evaluated, new rules can be added, and old rules may be removed or changed. Conducting these iterations will require the co-operation of a number of experts within the EW domain.

The Prolog program returns every feasible solution to any scenario described to it. The solutions are composed of combinations of manoeuvres and countermeasures. In performing the proper evasive actions the pilot will benefit from these combinations being prioritized. The prioritization can be done by either the increase in survivability given by the combinations, or by how easy they are to perform given the current status of countermeasures and direction of flight. On a graphical display the solutions can be presented in such a way that the person testing the system will see which threat each combination is mitigating.

The Prolog program does not take advantage of any knowledge about threats that will be encountered in the near future. Doing so will require a set of rules describing this to be formulated. Even though this formulation is not a trivial task, it can be introduced as part of the iterative improvement of the set of rules.

9.1.2 Bayesian Network

The time it takes for HUGIN to propagate evidence throughout the network depends on the number of nodes in the network. Section 5.3.2 explains that 36 combinations of states within four action nodes must be tested to find the one giving the highest survivability for a threat scenario. Since these combinations are mutually exclusive a single action node with 36 states can be constructed instead. This reduces the number of evidence propagations for each change in the BN from 36 to just one, which will result in a decrease of the total running time.

The BN model lacks the ability to determine the manoeuvre resulting in the best survivability. As it is any manoeuvre is preferred since deploying a countermeasure will almost always require a manoeuvre. If angles are introduced into the model the manoeuvre providing the best survivability can be found. The drawback of introducing angles is that the time it takes to propagate changes throughout the network is likely to increase.

The model includes only three types of countermeasures: flares, chaff, and jammer. If more countermeasures are included, and both new and existing countermeasures are described by a number of programs, the model will be closer to describing the situation for a fighter pilot. This will make it easier for e.g. a fighter pilot to perform an evaluation of the abilities of a BN approach.

Large parts of the BN model can be build using Structural Learning (SL). The learning feature offered by HUGIN seems to provide good results and it is thus a good candidate for performing the SL, although other tools for doing this may be investigated. Doing SL requires statistical data to be available. Statistical data from trials and real-life experience may be sufficient for the task, but the nature of these data makes them difficult to acquire. Synthetic data, e.g. provided by the Fly-In tool, may have the same characteristics as data from real aircraft. Gathering synthetic data can be a time-consuming task depending on the number of parameters included in the simulations. It will require more in-depth analysis to determine the parameters and the values of these to base the simulations on. Writing a Prolog program for delivering parts of these data may prove to be a viable approach; for instance in deciding proper angles for manoeuvres in conjunction with countermeasure deployments.

9.1.3 Mathematical Model

In Chapter 8 it was ascertained that solving the mathematical model using the GAMS/CPLEX approach is unlikely to produce usable results in real-time. Another use of the model is to find optimal solutions for problems that can be used in tuning parameters for the metaheuristic approach. Since parameter tuning is best done on problems similar to those which must be solved using the metaheuristic the implementation of the mathematical model must be able to solve these problems. In Section 6.4.3 it is estimated that a time frame may include up till 1500 time steps. Since finding the optimal solution to the CMOP with 20 time steps may take several hours with the GAMS/CPLEX approach, finding an optimal solution to a problem with 1500 time steps can not be done within reasonable time. Introducing more advanced OR techniques may be helpful in finding solutions to problems of this size faster, as relaxing the integer requirements may also prove to be beneficial.

The mathematical model is used to optimise the survivability for the aircraft flying a route defined by a set of IPs. A similar model may be developed to find IPs describing the route providing the optimal survivability. As route planning is a task that is most often carried out before a mission is initiated, finding a solution in real-time is no requirement.

9.1.4 Metaheuristic

As mentioned in Section 7.6 the initial solution to all but the first run of the Simulated Annealing may be a modification of the solution found at the previous run. The implementation of the metaheuristic may take advantage of this possibility. This initial solution might need to be modified if new threats have occurred, or if the battlefield scenario has otherwise changed.

In [40] the techniques for accelerating Simulated Annealing are classified into three categories: designing a faster algorithm by improving the cooling schedule, the neighbourhood, or the objective function; using hardware acceleration where time consuming parts of the algorithm is implemented in hardware; and finally parallelising the algorithm to take advantage of multiple processors. The Simulated Annealing implementation in this work has been evaluated using only a single neighbourhood definition and a single definition of the objective function. Experimenting with more implementations of these, as well as a variety of cooling schedules, may result in better performance within the fixed time interval given.

Since this work is of an exploratory nature implementing parts of the Simulated Annealing in hardware is considered beyond the scope. If a DSS is being implemented, and the choice of technique to use for this is a Simulated Annealing implementation, experiments using hardware acceleration need to be carried out. The aircraft computer running a DSS will probably be dedicated to this and having hardware running parts of the Simulated Annealing algorithm in this computer may thus be feasible. A disadvantage by having parts of the DSS implemented in hardware is that updating the DSS is likely to be more difficult.

According to [15, 40] parallelising the Simulated Annealing has been the subject of several studies. The parallelisation can be done in different ways, three of which are described here. The simplest way is to run an instance of the algorithm on each available processor. The instances would be stopped at the same time due to the fixed running time allowed, and the best solution is chosen. The immediate advantage of this approach is that it is relatively easy to implement since a minimum of synchronization between the processors is involved. If the instances are given the same initial solution and a very limited time to improve it, they may all explore the same part of the search space. It is not unlikely that the solutions found will be close, and there will little benefit from an increase in the number of available processors.

Introducing communication between the processors may result in better results. Each instance of the algorithm may search for solutions near the incumbent and report to the other instances when an accepted solution is found. This

solution will then be the incumbent for all instances in the continued search. This parallelisation will search larger parts of the search space during the fixed computation time allowed and the results is likely to be improved.

Having each processor working on a subproblem instead of the full problem may improve the solution found to each subproblem. For n processors the problem solved by the first processor may then contain only the first $1/n$ time steps, the next problem will describe the next $1/n$ time steps, and so on. The solutions found by each instance of the algorithm is reported in due time and every processor will have the fixed computation time allowed for solving only $1/n$'th of the problem. While the quality of the solutions to each of these subproblems will hopefully improve it can not be guaranteed that the combination of solutions to the subproblems constitute a good solution to the entire problem. Which parallelisations best suited for improving the results found using the Simulated Annealing may be the subject of further work.

With the implementation of the Simulated Annealing reported here the choices of cooling schedule, neighbourhood, and objective function may be subject for re-evaluation. Also the choice of metaheuristic may be re-evaluated. In Section 7.2.4 it is mentioned that the Simulated Annealing performs at its best if given enough time and if data describing the problem are uniform. Since the nature of the CMOP and the environment in which it needs to be solved offers neither optimal time conditions nor uniform data the choice of Simulated Annealing as metaheuristic may not be the best and other metaheuristics needs to be considered as well.

9.2 Testing with Flight Data

The four approaches have all been tested using synthetic data from scenarios fabricated to this purpose only. The Prolog program has been tested using warnings issued from the RWR and the MWS in well defined scenarios. These warnings are all caused by threats in the scenarios and the possibility of false warnings has not been exploited. Testing both the mathematical model and the metaheuristic implementations are done using a high-level description of scenarios where interaction with sensors on-board the aircraft is ignored. Testing with flight data instead of data constructed for test only may reveal flaws in the design of each of the approaches. It is recommended that this type of testing is carried out before further development is commenced.

9.2.1 Flight Simulator

At DDRE there has been some experience using the PC-based Falcon 4.0 flight simulator for supplying data to a tactical trainer [18]. These data can also be used for testing a PC-based DSS, where data may either be fed live to the system, or it may be recorded for later off-line data processing. With the live data feed the flight simulator will deliver flight data at a given frequency. The DSS then finds the combination of actions with the highest survivability given the flight data and these actions can be displayed on a monitor adjacent to the one showing the flight simulator.

If the DSS is run on the PC also running the flight simulator it might be difficult to meet the 200 milliseconds response time requirement. Therefore a two-PC setup is suggested; one running the flight simulator and one running the DSS. The connection between the computers can be established using an ordinary computer network cable. The delay in the transmission of data between the flight simulator and the DSS can be ignored since finding solutions to the scenario will still be the time consuming part of the process.

Falcon 4.0 will only supply the DSS with RWR data and the current amount of chaff and flares. While this is a drawback for testing the influence by MWS warnings and jammer data the connection between the Falcon 4.0 flight simulator and a DSS is still considered a good approach for performing flight data tests.

9.2.2 Flight Recorder Data

For some aircraft an on-board flight data recorder may deliver input to the test of a DSS. These data may consist of frequently collected aircraft positions and orientations, airspeed, sensor warnings, etc. An advantage with these data is that they carry the uncertainties experienced by the sensors aboard a real-life aircraft. A drawback is that the effect of actions suggested by the DSS will not be part of the data. While it may be possible to infer actions taken by the pilot, mapping these to possible actions suggested by the DSS may prove to be difficult.

9.2.3 Live Data Feed

The best way to test the actions suggested by a DSS will be to present the results to the fighter pilot during flight. Since the space in most fighter aircraft is limited it may be difficult to fit a DSS prototype into the cockpit. The test of a DSS does not require the aircraft to be a fighter aircraft, and it might as well be installed in a larger aircraft, e.g. in a military transport aircraft. Here data can be acquired from a MIL-STD-1553B (see Section 3.5.1) bus in much the same way as in a fighter aircraft[1]. A transport aircraft may be equipped with largely the same countermeasures as a fighter aircraft and although the speed and manoeuvrability of the transport aircraft is different from those of the fighter aircraft it is still possible to measure the effects of actions suggested by the DSS.

To ensure input data from on-board warning system such as RWR and MWS the testing of the system needs to be performed flying over threats. Flying a transport aircraft over enemy territory can not be regarded as optimal conditions for doing countermeasure or DSS tests. For this the NATO Air Force Armaments Group organise a series of tests with the participation of equipment and aircraft from different NATO countries. Here the aircraft overfly real or simulated threats to test the effect of aircraft countermeasures. Two series are organised: trial MACE covers RF based threats and trial EMBOW covers testing against electro-optical systems including IR threats [43].

9.3 Other Techniques

Other techniques than the four described in this work may be viable for the construction of a DSS for fighter pilots. Some of the candidates that were discussed during the work are described in this section.

9.3.1 Constraint Logic Programming

Constraint Logic Programming (CLP) is a combination of two declarative programming paradigms: logic programming and constraint solving. With logic programming variables have the scope of the current predicate only. In CLP constraints are introduced to represent relations between variables throughout

[1]Not all military transport or fighter aircraft are equipped with the MIL-STD-1553B databus. Acquiring data from a non-standard databus may be inconvenient but still possible.

a given domain such as trees and sets. Handling these constraints is done using a programming language that adds to the functionality of Prolog [56]. Generally there is a strong connection between Prolog and CLP, and many Prolog interpreters offers some degree of CLP, while compilers dedicated to CLP will often be able to compile Prolog programs.

CLP can be used for adding functionality to the Prolog program described in Chapter 4, and according to e.g. [56] it may enhance both the productivity of software development and software maintainability. Several CLP systems can be investigated for this, e.g. the B-Prolog system introduced in Chapter 4 and the ECL^{i}PS^e system [2].

9.3.2 Artificial Neural Net

Using an Artificial Neural Net (ANN) a number of neurons can represent the same set of variables as introduced in the BN approach. While the results with a BN is found by propagating probabilities throughout the entire network, the results from a ANN is found using a number of functions integrated with the relevant sequences of neurons from the input to the output.

Updating the dependency tables in the BN has shown to be a disadvantage with this approach. Learning the structure and dependency tables using SL requires representative data for all combinations of states within the variables of the BN. If only parts of this set of data is available SL will come short of providing a usable BN. If a learning ANN is used instead the setup of functions and weighs within the network will be adjusted according to the data it can be trained with. If this training is done using e.g. data from a flight simulator it may show that data representing all combinations of states are not necessary to train the ANN to provide good solutions.

In [38] an ANN is taught to play poker without knowing any expert strategies. The ANN here showed the ability to play and win a poker game where knowledge of cards held by the opponents was unavailable. A similar teaching scheme can be applied to an ANN for finding actions to be performed by a pilot when ground-based threats emerge. Here the opponent represent the enemy, the card held by the opponent are hidden to the system in much the same way as the number and positions of threats, and finally information about approaching missiles can be compared to the revealed community cards in Texas Hold'em poker.

Exploring the ANN approach can be done by either developing relevant software or by using commercial software such as the Neural Net Toolbox in MATLAB.

9.3.3 Stochastic Programming

Stochastic Programming is a framework for modelling uncertainty involved in optimisation problems. The mathematical model described in Chapter 6 describes a deterministic optimisation problem where the parameters describing both the threats and the effects of countermeasures are known. In a real-world scenario each of these parameters can be subject to uncertainty and a Stochastic Programming approach will include these uncertainties when suggesting a combination of actions to the pilot. As with the BN it is not a trivial task to determine the uncertainties to include in the model.

Chapter 10

Conclusion

Describing the problematics involved in deciding the optimal response to enemy threats is a rather complex task that involves many facets. For each of the models developed in this work a large part of these facets are left out while others have been simplified to obtain a working model. For domain experts within the field of EW these omissions and simplifications may seem unnecessary and the solutions found using the models may be too simple to have any practical value.

Early on in the work with this project it was decided to focus the work on ground-based threats only. This was done to eliminate the number of facets involved in the models, and the assumption was that "raising" the threats into the air was probably only a matter of introducing an altitude to the description of each threat. During the work it has been shown that besides modelling the problem another critical issue was to obtain responses from the systems in real-time. While this is a cornerstone in the requirements for a DSS finding mitigating actions to ground-based threats it is even more important if the threats are airborne. Here the pilot alone does not determine e.g. the distance to an enemy aircraft and any manoeuvres performed to avoid missiles may be countered by the enemy. Since there may be little risk of a fighter aircraft engaging in a dogfight, focussing on ground-based threats only is still considered a good decision.

A large part of the time spent on this project has been used on the construction

of data for the development and testing of the models. This construction has been necessary since real-world data have not been available. Although the constructed data have some resemblance to data that may be gathered in real-world scenarios they must be considered with great caution. One must exercise great care when executing decisions based on the models build or tested using constructed data since these data may give a poor representation of the world surrounding a real-world aircraft. If representative data had been available throughout the project less time had been spent on the construction of data and data generating models, and more time had been dedicated to improving the results of the four approaches. It is likely that this had improved the usability of the systems developed.

With the approaches chosen for this work it has been shown that it is possible to find usable results to different formulations of the situation for a fighter pilot. Whether one of these approaches, or perhaps a combination of them, will provide the best results possible has not been shown. Determining the best approach for the development of a real-world DSS requires further work.

Appendix A

Threats

A.1 Guidance Systems

A number of guidance systems are described in Table A.1. To each guidance system an illustration of the parties and radiations involved is given.

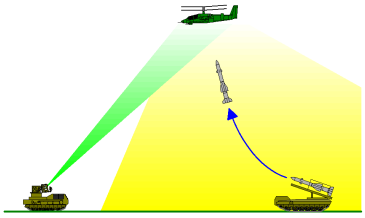
Semi-Active Radar (SAR) Missile Guidance In a semi-active radar guided system, the nose of the missile contains a radar receiver which receives radar reflections from a target illuminated by an associated target tracking radar platform. In the illustration the missile is fired at a helicopter from a ground based vehicle. The radar radiation (green) is emitted from a radar platform, and echoed off the aircraft (yellow).	

Table A.1: Description of guidance systems. *Continues...*

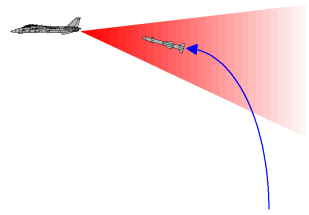
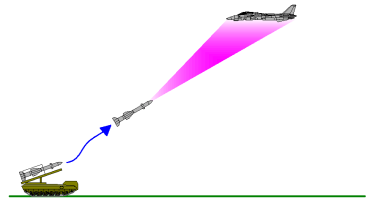
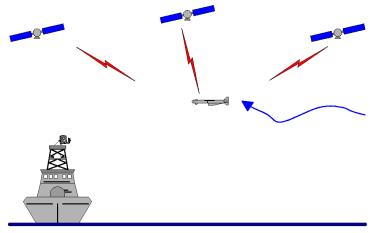
Infrared (IR) Missile Guidance In an IR guidance system the nose of the missile contains a sensor that is sensitive to the IR portion of the electromagnetic spectrum. This sensor is capable of detecting the IR radiation emitted from a target. IR seekers can be of two types, homing or imaging. This type of missile is a "fire-and-forget" system. In the illustration the red radiation represents the IR radiation emitted by the aircraft.	
Active Radar Missile Guidance In an active radar guidance system the missile contains a complete radar system which transmits radar radiation and receives radar reflections from the target. This type of missile is a "fire-and-forget" system. The radiation shown in the illustration is both the transmitted and received radar radiation.	
Inertial Navigation System (INS) Guidance In an Inertial Navigation System (INS)/Global Positioning System (GPS) system, the missile is launched and navigates to the designated target based on its launch location and target location. The course is computed using flight data of the missile and/or GPS data. Once in the target area, this type of system often has an active radar or IR imaging homing mode to increase the accuracy of the weapon. In the illustration the target is a ship.	

Table A.1: Description of guidance systems. *Continues...*

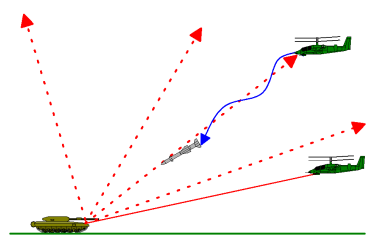
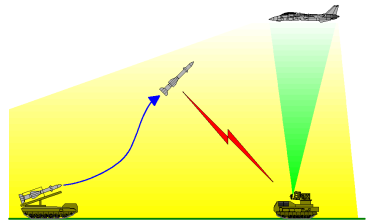
Electro-Optical (EO) Missile Guidance In an Electro-Optical (EO) guidance system, the nose of the missile contains a seeker that is sensitive to the IR or optical portion of the electromagnetic spectrum. The missile guides to the target by tracking the EO reflections off a target. Target illumination can be accomplished by the launching platform or secondary source. In the illustration the target is illuminated by the helicopter at low altitude while the missile is launched from the second helicopter.	
Track-Via-Missile (TVM) Guidance A TVM system is very similar to the SAR system. A receiver in the nose of the missile receives radar reflections from the target and downlinks the data to a ground station. Course corrections are computed at the ground station based on local radar data on the target and the downlinked data from the missile. Once computed, course corrections are uplinked to the missile enabling a high degree of accuracy in course intercept between the target and missile. The green radiation in the illustration represents the radiation emitted by a ground based radar system. The yellow is the radiation echoed off the aircraft. The missile is guided towards the aircraft.	

Table A.1: Description of guidance systems. *Continues...*

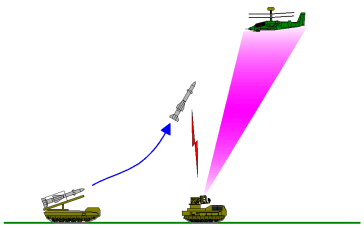
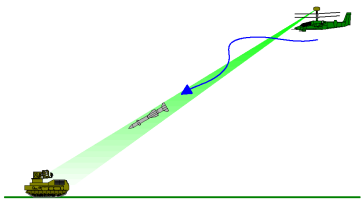
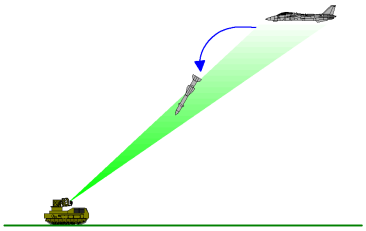
Command Guidance In a command guidance system, the missile is datalinked via radio to typically a ground station. The ground station computes trajectory corrections based on radar/IR/optical inputs from both the target and missile. These course corrections are then uplinked to the missile to ensure intercept. In some systems, both the target and missile must stay within the launcher's field of view - this type of systems is called a Command-Line-of-Sight (CLOS) system.	
Beam Rider Guidance In a beam riding missile system, the missile contains a radar receiver in its tail, and "rides" the radar beam to the target. In the illustration the missile is fired and guided from the helicopter.	
Anti-Radiation Missile (ARM) Guidance In an ARM based guidance system, the missile contains a radar receiver, which homes in on electromagnetic energy emitted by the target, i.e. its own radars or jamming equipment. In the illustration the missile is guided towards the ground based radar.	

Table A.1: Description of guidance systems. (Source: RIA4 Missile Guidance Series, Set 1)

A.2 Surface-to-Air Missile Reference Guide

The table below show some known threats and their association to different radars.

Name	Type	Length	Guidance	Max speed	Max range	Min range	Max alt.	Min alt.	Launcher
SA-2	Strategic med -high	20'0"	Command	M 3.5	19-27 NM	3.5 NM	90,000'	295'	Single rail
SA-3	Strategic low -med	20'0"	Command	M 3.5	14 NM	1.3 NM	60,000'	150'	Double or four rail
SA-5	Strategic med -high	35'3"	SARH	M 4+	170 NM	25 NM	100,000'	984'	Single rail
SA-6	Strategic low - med	19'0"	SARH	M 2.5	13 NM	2.0 NM	47,000'	100'	TEL with 3 missiles
SA-8	Strategic low - med	10'6"	Command	M 2.0	6.6 NM	0.9 NM	42,600'	33'	TELAR with 6 missiles
SA-10	Strategic low, high	23'4"	Command or QAS/TVM	M 6.0	49 NM	2.6 NM	88,500'	80'	TEL with 4 missiles
SA-11	Mobile low - med	18'4"	SARH	M 3.0	16 NM	1.6 NM	72,000'	50'	TELAR with 4 missiles
SA-12	Mobile low - high	26'11"	Inertial Command SARH		43.2 NM				TELAR with 4 missiles
SA-12B	Mobile low - high	34'5"	Inertial Command SARH		81 NM				TELAR with 2 missiles
SA-13	Mobile low	7'3"	IR	M 1.5	4.4 NM	0.25 NM	10,500'	33'	TELAR with 4 missiles
SA-15	Mobile low	11'6"	Command and active	M 2.5	6.5 NM	1.3 NM	19,700'	33'	TLAR with 3 missiles
SA-16	MANPAD low	4'7"	IR	M 1.8+	3.5+ NM	0.3-33 NM	18,000'	30'	Shoulder fired
SA-19	Mobile low	8'2"	ACLOS or SACLOS	Hypersonic	4.3 NM	1.3 NM			TELAR with 3 missiles
ASPID	Mobile low - med	12'3"	SARH	M 2.5	8.1 NM		20,000'	50'	TELAR with 4 - 8 missiles
CROTALE	Mobile low - med	9'6"	Command and optical	M 3	5.5 NM	0.3 NM	18,000'	50'	TELAR with 4 missiles
I-HAWK	Strategic low - med	16'6"	SARH	M 2.5	21.6 NM	0.9 NM	48,000'	90'	Launcher with 4 missiles
RAPIER	Point low	7'4"	Command and optical	M 2+	4 NM	0.3 NM	10,000'	50'	TEL with 4 missiles
RBS-70	MANPAD low	4'2"	Laser beam rider	M 1+	3.3 NM	0.1 NM	13,000'	LOS	MANPAD with 2 missiles
ROLAND	Mobile low - med	8'6"	Command and optical	M 1.6	3.5 NM	0.4 NM	18,000'	66'	TELAR with 2 missiles
STINGER	MANPAD low	5'0"	IR/UV	M 2.2	2.5 NM	0.1 NM	12,500'	LOS	Shoulder fired

Appendix B

The Prolog Program

B.1 Rules

1. Probable threats are determined pre-mission.
2. Environment hostility depends on the number of anticipated threats.
3. Number of anticipated threats is based on intelligence.
4. A transport aircraft experience hostile environment during take-off and landing.
5. Data is collected real-time via datalink and warning systems.
6. The breaklock zones for chaff are at 4 o'clock and at 8 o'clock.
7. In case of a missile warning there is no time to deploy the towed decoy.
8. Dispense chaff and flares only when the distance to the threat is right.
9. For chaff the distance to the threat should be less than 5 km and more than 500 m.
10. If flares are to be used reactively, the distance to the threat should be less than 1 km and more than 100 m.

11. If flares are to be used pre-emptively they should be dispensed as soon as possible, regardless of any distance.
12. Flares are to be used if the aircraft is in very hostile environment and flying in an altitude below 1000 m.
13. Chaff has no effect against a Doppler radar. Use the jammer instead.
14. At altitudes above 20,000 ft MANPADS is not posing a threat. Use only chaff.
15. At altitudes below 1,000 ft RF guided missiles are not posing a threat.
16. Chaff and flares can be used as responses to missile warnings.
17. For timing issues jammer and towed decoy can not be used as responses to warnings, unless they are already in use.
18. If the MWS indicates a missile in a given direction, and the RWR does not, the missile is IR guided.
19. Dispense flares if an approaching missile is IR guided.
20. If both the MWS and the RWR indicate a missile in a given direction, the missile is RF guided.
21. Dispense chaff if an approaching missile is RF guided.
22. If a missile is approaching select the proper countermeasure and manoeuvre.
23. Manoeuvres are determined by the breaklock zones of the proper countermeasure.
24. The plumes from other aircraft, e.g. wingman and coalition forces, may cause false alarms by the MWS.
25. The positions of other aircraft may be continuously updated, e.g. via datalink.
26. When the jammer is in 'auto' mode, it will jam the RF sources detected.
27. Jamming will not commence before the power of the RF source is above detection level.
28. The jammer mode is set to 'auto' when the aircraft fly over hostile environment.
29. When the jammer is jamming it may influence the RWR and jammers of other aircraft.

30. The jammer will reveal the position of the aircraft.
31. When flying at high altitudes, the surrounding air is thinner than when flying at low altitudes.
32. To maintain the effect of flares, when flying in thin air, the amount of flares should be increased.
33. Chaff and flare programs depend on the remaining amount of chaff and flares.
34. Chaff and flare programs depend on the estimated TTG.
35. Flare programs depend on the type of an approaching missile and the altitude.
36. The types of missiles to anticipate depend on intelligence, and are given as pre-mission support.
37. The jammer and the towed decoy should not be used simultaneously, unless they cover different threats.
38. The use of a towed decoy may limit the aircraft manoeuvrability.
39. The RF countermeasure to use may depend on table lookups.
40. The MWS can not distinguish between different types of IR threats

B.2 dss.pro

```
%--------------------
% Include additional files.
%--------------------
:-include('threats.pro').
:-include('cm.pro').
:-include('mission.pro').
:-include('current.pro').
:-include('warnings.pro').
:-include('util.pro').


%--------------------
% The main entry. 'go' will return the total execution time,
% measured in miliseconds.
%--------------------
go :-
    statistics(runtime,[Start|_]),
```

```
    what_to_do,
    statistics(runtime,[End|_]),
    T is End-Start,
    nl, write('Execution time is '),
    write(T), write(' milliseconds'), nl.


%--------------------
% Warnings are handled and responses to the environment are
    found.
%--------------------
what_to_do :-
    %--- Handle warnings
    (
        setof((Warner, WarnInfo), warning(Warner, WarnInfo),
            Warnings),
        setof(_,(memberof(W, Warnings), handle_warning(W)),
            _), !
        ;
        true
    ),

    %--- Environment responses
    (
        ir_mode(preemptive),
        altitude(Alt),
        Alt < 6000,
        (
            available(flares),
            not(cm_has_effect(flares)),
            write_cm(flares, flaresdef), !
            ;
            available(dircm),
            not(cm_has_effect(dircm)),
            write_cm(dircm, auto), !
        )
        ;
        true
    ),(
        rf_hostility(high),
        altitude(Alt),
        Alt > 300,
        (
            cm_has_effect(jammer), !
            ;
            cm_has_effect(towed_decoy), !
            ;
```

```
            cm_has_effect(chaff), !
            ;
            available(jammer),
            write_cm(jammer, auto), !
            ;
            available(towed_decoy),
            write_cm(towed_decoy, auto), !
            ;
            available(chaff),
            write_cm(chaff, chaffdef), !
        )
        ;
        true
    ).


%--------------------
% Find actions to each warning, and write them to the screen.
%--------------------
handle_warning((Sensor, (Angle, WarnData))) :-
    (
        Sensor = rwr
        ;
        Sensor = mws,
        not(friend(Angle))
    ),
    recommend_action((Sensor, (Angle, WarnData)), Cm, Man,
        Prog),

    %--- Write results
    write_threat((Sensor, (Angle, WarnData))),
    write_manoeuvre(Man),
    write_cm(Cm, Prog).


%--------------------
% Recommend action (countermeasure, manoeuvre, and programme)
%--------------------
recommend_action(Warning, Cm, Man, Prog) :-
    recommend_cm(Warning, Cm),
    recommend_man(Warning, Cm, Man),
    prog(Cm, Prog).


%--------------------
% Which countermeasures should be recommended to mitigate
% threats at Angle?
```

```
%-------------------
recommend_cm((_,(Angle,_)), Cm) :-
    approp_list(Angle, Cms),
    memberof(Cm, Cms),
    available(Cm),
    not(cm_has_effect(Cm)),
    mitigates(Phys, Cm),
    not(safe_altitude(Phys)),
    (
        %-- If a distance to the threat is known,
        %-- is it then a lethal distance?
        warning(mws, (Angle, Dist)),
        lethal_dist(Cm, Dist), !
        ;
        true
    ).


%-------------------
% Recommend a manoeuvre for the threat and countermeasure.
%-------------------
recommend_man((_,(ThreatAngle,_)), Cm, (Direction, Steps)) :-
    breaklock(Cm, BreakAngle),
    manoeuvre(BreakAngle, ThreatAngle, Direction, Steps).


%-------------------
% When flying at a safe altitude, certain types of
% guidance does not pose a threat
%-------------------
safe_altitude(ir) :-
    altitude(Alt),
    Alt > 6000.
safe_altitude(uv) :-
    altitude(Alt),
    Alt > 6000.
safe_altitude(rf) :-
    altitude(Alt),
    Alt < 300.


%-------------------
% Lethal distances and safe altitudes for different
% guidance systems
%-------------------
lethal_dist(chaff, Dist) :-
    Dist > 500,
```

```
    Dist < 5000.
lethal_dist(flares, _) :-    % Always in lethal distance
    ir_mode(preemptive).
lethal_dist(flares, Dist) :-
    ir_mode(reactive),
    Dist > 100,
    Dist < 1000.


%--------------------
% Make list of appropriate countermeasures.
%--------------------
approp_list(Angle, Cms) :-
    setof(Cm, proper_cm(Angle, Cm), Cms).


%--------------------
% Find appropriate countermeasures.
%--------------------
proper_cm(Angle, jammer) :-
    (
        warning(rwr, (Angle, _)),
        warning(mws, (Angle, _)),
        jammer_mode(auto)
    );(
        warning(rwr, (Angle, _)),
        warning(mws, (DiffAngle, _)),
        Angle \== DiffAngle
    );(
        warning(rwr, (Angle, _)),
        not(warning(mws, (_, _)))
    ).

proper_cm(Angle, towed_decoy) :-
    (
        warning(mws, (Angle, _)),
        warning(rwr, (Angle, _)),
        decoy_mode(deployed)
    );(
        warning(rwr, (Angle, _)),
        warning(mws, (DiffAngle, _)),
        Angle \== DiffAngle
    );(
        warning(rwr, (Angle, _)),
        not(warning(mws, (Angle, _)))
    ).
```

```
proper_cm(Angle, chaff) :-
    warning(rwr, (Angle, Threat)),
    not(doppler(Threat)).

proper_cm(Angle, dircm) :-
    (
        warning(rwr, (DiffAngle, _)),
        warning(mws, (Angle, _)),
        Angle \== DiffAngle,
        dircm_mode(auto)
    );(
        not(warning(rwr, (_, _))),
        warning(mws, (Angle, _)),
        dircm_mode(auto)
    ).

proper_cm(Angle, flares) :-
    warning(mws, (Angle, _)),
    not(warning(rwr, (Angle, _))).


%-----------------
% Determine IR mode.
%-----------------
ir_mode(preemptive) :-
    altitude(Alt),
    Alt < 1000,
    ir_threat(severe), !.
ir_mode(preemptive) :-
    ac_type(transport),
    fly_mode(take_off), !.
ir_mode(preemptive) :-
    ac_type(transport),
    fly_mode(landing), !.
ir_mode(reactive).


%-----------------
% If RF threats are known, the environment is hostile.
%-----------------
rf_hostility(high) :-
    threat_probable(T),
    guidance(T, B),
    phys_guidance(B, rf), !.
rf_hostility(low).
```

```
%-------------------
% Estimate IR threat status.
%-------------------
ir_threat(none) :-
    count_ir_threats(N),
    N = 0, !.
ir_threat(moderate) :-
    count_ir_threats(N),
    N > 0, N < 3, !.
ir_threat(severe) :-
    count_ir_threats(N),
    N > 2.


%-------------------
% Count IR threats to estimate the environment hostility.
%-------------------
count_ir_threats(N) :-
    findall(Threat,
        (threat_probable(Threat),
        guidance(Threat, Guidance),
        phys_guidance(Guidance, ir)),
    Threats),
    count(N, Threats).
```

B.3 `util.pro`

```
%-------------------
% Writing the results to the screen.
%-------------------
write_threat((Sensor, (Angle, WarnData))) :-
    (
        Sensor = rwr,
        write('RWR: '),
        write(WarnData),
        write(' at '),
        write(Angle),
        nl, !
    );(
        write('MWS: Missile at '),
        write(Angle),
        write(', distance '),
        write(WarnData),
        write(' meters'),
        nl
    ).
```

```
write_manoeuvre((Direction, Steps)) :-
    (
        Steps == 0,
        write('Stay on course'),
        nl, !
    );(
        write('Turn '),
        write(Direction),
        write(', '),
        write(Steps),
        write(' step(s)'),
        nl
    ).

write_cm(Cm, Prog) :-
    write('Use '),
    write(Cm),
    write(', program '),
    write(Prog),
    nl,
    nl.


%-----------------
% Angles.
%-----------------
turn_right(one_o_clock,    two_o_clock   ).
turn_right(two_o_clock,    three_o_clock ).
turn_right(three_o_clock,  four_o_clock  ).
turn_right(four_o_clock,   five_o_clock  ).
turn_right(five_o_clock,   six_o_clock   ).
turn_right(six_o_clock,    seven_o_clock ).
turn_right(seven_o_clock,  eight_o_clock ).
turn_right(eight_o_clock,  nine_o_clock  ).
turn_right(nine_o_clock,   ten_o_clock   ).
turn_right(ten_o_clock,    eleven_o_clock).
turn_right(eleven_o_clock, twelve_o_clock).
turn_right(twelve_o_clock, one_o_clock   ).

turn_left(X, Y) :-
  turn_right(Y, X).


%-----------------
% Manoeuvres.
%-----------------
```

```
manoeuvre_left(Same, Same, Steps) :-
  Steps is 0.
manoeuvre_left(From, To, Steps) :-
  turn_left(From, NewAngle),
  manoeuvre_left(NewAngle, To, MoreSteps),
  Steps is MoreSteps + 1.

manoeuvre_right(Same, Same, Steps) :-
  Steps is 0.
manoeuvre_right(From, To, Steps) :-
  turn_right(From, NewAngle),
  manoeuvre_right(NewAngle, To, MoreSteps),
  Steps is MoreSteps + 1.

manoeuvre(From, To, Direction, Steps) :-
  manoeuvre_left(From, To, StepsLeft),
  manoeuvre_right(From, To, StepsRight),
    (StepsLeft < StepsRight, Steps is StepsLeft, Direction =
        left, !;
    Steps is StepsRight, Direction = right, !).


%-----------------
% List functions.
%-----------------
%--- Number of members
count(0, []) :- !.
count(N, [_|Tail]) :-
    count(N1, Tail),
    N is N1 + 1.

%--- Membership
memberof(X,[X|_]).
memberof(X,[_|Tail]) :- memberof(X, Tail).
```

B.4 cm.pro

```
%-----------------
% Break-lock zones
%-----------------
breaklock(chaff, four_o_clock).
breaklock(chaff, eight_o_clock).
breaklock(jammer, one_o_clock).
breaklock(jammer, five_o_clock).
breaklock(jammer, six_o_clock).
breaklock(jammer, seven_o_clock).
```

```
breaklock(jammer, eleven_o_clock).
breaklock(jammer, twelve_o_clock).
breaklock(Cm, _) :-
    (Cm = chaff, !, fail) ;
    (Cm = jammer, !, fail) ;
    true.


%--------------------
% Countermeasure programs
%--------------------
%--- Flare programs

prog(flares, flares01) :-
    flares_left(FL),
    FL > 5,
    altitude(Alt),
    Alt < 300, !.

prog(flares, flares02) :-
    flares_left(FL),
    FL > 7,
    altitude(Alt),
    Alt < 500, !.

prog(flares, flaresdef) :-
    flares_left(FL),
    FL > 7, !.

prog(flares, none) :- fail, !.

%--- Chaff programs
prog(chaff, chaff01) :-
    chaff_left(CL),
    CL > 6,
    altitude(Alt),
    Alt < 500,
    warning(rwr,(_,sa2)), !.

prog(chaff, chaff01) :-
    chaff_left(CL),
    CL > 6,
    altitude(Alt),
    Alt < 1000,
    warning(rwr,(_,sa6)), !.

prog(chaff, chaffdef) :-
```

```
    chaff_left(CL),
    CL > 5, !.

prog(chaff, _) :- fail, !.

%--- Default (at the end)
prog(_,default).


%-------------------
% Are the countermeasures currently mitigating?
%-------------------
cm_has_effect(towed_decoy) :-
    decoy_mode(deployed).

cm_has_effect(jammer) :-
    jammer_mode(auto).

cm_has_effect(dircm) :-
    dircm_mode(auto).

cm_has_effect(chaff) :-
    chaff_disp(Time),
    Time < 3.

cm_has_effect(flares) :-
   flares_disp(Time),
   Time < 1.
```

B.5 threats.pro

```
%-------------------
% Missile systems
%-------------------
guidance(sa2, command).
guidance(sa3, command).
guidance(sa5, sarh).          % Doppler
guidance(sa6, sarh).
guidance(sa8, command).
guidance(sa10, command).      % Doppler
guidance(sa10, qas_tvm).
guidance(sa11, sarh).
guidance(sa12, inertial).
guidance(sa12, command).
guidance(sa12, sarh).
guidance(sa12b, inertial).
```

```
guidance(sa12b, command).
guidance(sa12b, sarh).
guidance(sa13, ir).
guidance(sa15, command).
guidance(sa15, active).
guidance(sa16, ir).
guidance(sa18, ir).
guidance(sa19, aclos).
guidance(sa19, saclos).
guidance(aspid, sarh).
guidance(crotale, command).
guidance(crotale, optical).
guidance(ihawk, sarh).
guidance(rapier, command).
guidance(rapier, optical).
guidance(rbs70, laser_beam_rider).
guidance(roland, command).
guidance(roland, optical).
guidance(stinger, ir).
guidance(stinger, uv).
guidance(manpads, ir).


%-------------------
% RF threats based on Doppler.
%-------------------
doppler(sa5).
doppler(sa10).


%-------------------
% The electromagnetic band used by the guidance systems.
%-------------------
phys_guidance(command, rf).
phys_guidance(sarh, rf).
phys_guidance(qas_tvm, rf).
phys_guidance(inertial, rf).
phys_guidance(ir, ir).
phys_guidance(aclos, ir).
phys_guidance(saclos, ir).
phys_guidance(optical, uv).
phys_guidance(laser_beam_rider, uv).
phys_guidance(uv, uv).


%-------------------
% Countermeasure mitigating in certain bands.
```

```
%-------------------
mitigates(ir, flares).
mitigates(ir, dircm).
mitigates(rf, chaff).
mitigates(rf, jammer).
mitigates(rf, towed_decoy).
mitigates(uv, flares).
mitigates(uv, dircm).
```

B.6 mission.pro

```
%-------------------
% A/C type
%-------------------
ac_type(fighter).


%-------------------
% Countermeasures available
%-------------------
available(flares).
available(chaff).
available(towed_decoy).
available(jammer).
available(dircm).


%-------------------
% Probable threats
%-------------------
threat_probable(sa2).
threat_probable(sa3).
threat_probable(sa10).
threat_probable(sa13).
threat_probable(sa13).
threat_probable(sa13).
threat_probable(sa13).
threat_probable(sa13).
```

B.7 current.pro

```
%-------------------
% Ownship data
%-------------------
```

```
altitude(1600).
fly_mode(cruise).
flares_left(10).
chaff_left(10).


%-------------------
% Countermeasure modes
%-------------------
decoy_mode(not_deployed).    % deployed or not_deployed
jammer_mode(auto).           % auto, receive, or off
dircm_mode(off).             % auto, receive, or off
chaff_disp(100).             % Seconds since chaff was
    dispensed
flares_disp(100).            % Seconds since flares were
    dispensed


%-------------------
% Friendly aircraft
%-------------------
%friend(one_o_clock).
%friend(five_o_clock).
```

B.8 warnings.pro

```
%-------------------
% Warning descriptions
%-------------------
warning(mws, (six_o_clock, 500)).
warning(mws, (nine_o_clock, 500)).
warning(rwr, (nine_o_clock, sa5)).
```

Appendix C

Survival Score

C.1 Constructing a score system

In medicine score systems like Glasgow Coma Score [50] and Apgar Score [8] are used to give the medical staff a fast overview of the conditions of a patient. The advantage of score systems is that they will get this overview without the labour and time it will take to do a more detailed examination. The score systems are often used to make decisions on the treatment of the patient when there is no time to do more examinations. A score may include scores from other score systems.

Constructing a score system for the survivability of a fighter aircraft has much the same use as a medical score system. A higher score indicates a higher survivability for the aircraft. Optimising the score is another way of giving the aircraft the best chances of surviving. A difference is the use of simplicity: a medical score is often fast calculated by the medical staff based on a few observations, and without any tools for doing the calculations. It therefore has to be relatively simple and easy to calculate. The survival score can be calculated with more complex parts since it will be calculated automatically.

C.1.1 Contributions to the score

The list below shows some of the aspects to be taken into account when calculating a survival score:

1. Threats contribute with a negative value. Probability of the threat is used for weighing the value.

2. Multiple threats increase the threat value. The increase is not proportional to the number of threats since applied countermeasures may influence more than one threat.

3. Applying proper countermeasures should add a positive number of the same magnitude as that of the threat being treated. Difference between current aspect angle and the preferred angle is used as weight.

4. The score can have a temporal aspect, depending on the recent history. If proper countermeasures have already been applied, and the effect of these has not yet been registered, the threat contributes with a numerical smaller value.

5. The score may also depend on the remaining amount of chaff and flares. Having the possibility of using a proper countermeasure adds one value, actually using it adds another.

6. Using the wrong countermeasure adds a penalty value, since this might decrease the survivability later on in the mission.

7. The probability of emerging threats, depending on e.g. the territory, will also have an influence on the score. This may make the altitude relevant.

Some of the contributing parts do not need to be included every time the score is calculated. An example of this is the part concerning the altitude. Changing the altitude is a relatively slow process, and even though a change in altitude is in progress it is fair to assume that this part should not be considered more often than once every two seconds or even more infrequently.

C.1.2 Formalizing the score

S	The survivability score
$K = \{IR, RF, ...\}$	The domains of threat to consider
$P(k), k \in K$	The probability of a threat k. Depends on sensor output
$a_k \in \{0, 1\}$	The applicability of the score from the k'th domain
s_k	Score for the k'th domain
$L_k \in \{0, 1\}$	Countermeasures loaded/present
$P(A_k)$	The probability of the countermeasure being applied
$P(W_k)$	The probability of the countermeasure working as intended
$T_k \in \mathbb{R}_-$	Maximal threat value
$C_k \in \mathbb{R}_+$	Maximal countermeasure value
$p_k(t)$	The penalty for using countermeasures at time t

The survivability score can be calculated as:

$$
\begin{aligned}
S &= \sum_{k \in K} a_k \cdot s_k && \text{(C.1)} \\
s_k &= -P(k) \cdot T_k && \text{(C.2)} \\
&\quad +L_k \cdot P(A_k) \cdot P(W_k) \cdot C_k && \text{(C.3)} \\
&\quad -p_k(t) \cdot P(A_k) && \text{(C.4)}
\end{aligned}
$$

In the above (C.2) is the threat part of the survival score, (C.3) is the countermeasure part, and (C.4) is the penalty part.

C.2 Optimising the score

Let us rewrite the equation (C.2) – (C.4) as:

$$s_k = -\mathcal{T}_k + \mathcal{C}_k - \mathcal{P}_k \tag{C.5}$$

It is obvious from equation (C.5) that the maximum value for s_k is found when the threat part ($\mathcal{T}_k$) and the penalty ($\mathcal{P}_k$) are small and when the countermeasure ($\mathcal{C}_k$) is high. Of these the pilot can only directly influence the countermeasure and penalty parts. If no threat is present the penalty part should have a value numerical larger than that of the countermeasure part.

$$\mathcal{T}_k = 0 \Rightarrow \mathcal{P}_k > \mathcal{C}_k$$
$$\mathcal{T}_k \neq 0 \Rightarrow \mathcal{P}_k \ll \mathcal{T}_k$$

C.3 Further work

In deciding on the formulation of the survival score one must consider the following:

1. More countermeasures to same kind of threat
2. Threats without countermeasures
3. When is a score system "good enough"?
4. How bad can a full description be, before it is too bad?

Appendix D

The GAMS Program

D.1 tempasp.gms

```
* use: gams tempasp u1="flight.dat"

$eolcom //
option iterlim  = 999999999;     // avoid limit on iterations
option reslim   = 86400;         // time limit for solver in
    sec.
option optcr    = 0.;            // gap tolerance
option solprint = ON;            // include solution print in
    .lst file
option limrow   = 100;           // limit number of rows in
    .lst file
option limcol   = 100;           // limit number of columns in
    .lst file
//-----------------------------


////////////////////
Sets
    cm      'Countermeasures'   / 'jammer', 'decoy', 'chaff',
        'noCm'/
    ang     'Angles of Attack'  / 0 * 360 /
```

```
    dist     'Distances'          / 0 * 100 /
    t        'Time steps'
    thr      'Threats'
    ;


////////////////////
Parameters
    redAlpha(cm,ang)     'Reduction of lethality (Alpha)'
    redRho(cm,dist)      'Reduction of lethality (Rho)'
    flyAng(t,thr)        'Angles during flight'
    flyDist(t,thr)       'Distances during flight'
    lethality(t,thr)     'Lethality of threats during flight'
    probThreat(t,thr)    'Probability of threat'
    Rjmr(t,thr)          'Reduction by jammer'
    Rjmra(t,thr)         '... angle'
    Rjmrd(t,thr)         '... distance'
    Rdec(t,thr)          'Reduction by decoy'
    Rdeca(t,thr)         '... angle'
    Rdecd(t,thr)         '... distance'
    Rchf(t,thr)          'Reduction by chaff'
    Rchfa(t,thr)         '... angle'
    Rchfd(t,thr)         '... distance'
    minDist(t,thr)       'Distance to threat (less than 200)'
    ;

$include "reductions.dat"    // Include reductions
$include "%gams.user1%"      // Include angles, distances, and
    lethalities

    ///////
    // Find the reductions
    alias(t,    t1);
    alias(dist, d1);
    alias(ang,  a1);
    alias(thr,  thr1);

    loop(t1,
        loop(thr1,
            loop(d1,
                if(ord(d1)-1 = flyDist(t1,thr1),
                    Rjmrd(t1,thr1) = redRho('jammer',d1)
                );
            );
            loop(a1,
                if(ord(a1)-1 = flyAng(t1,thr1),
                    Rjmra(t1,thr1) = redAlpha('jammer',a1)
```

```
            );
        );
    );
);
Rjmr(t,thr) = Rjmrd(t,thr) * Rjmra(t,thr);

loop(t1,
    loop(thr1,
        loop(d1,
            if(ord(d1)-1 = flyDist(t1,thr1),
                Rdecd(t1,thr1) = redRho('decoy',d1)
            );
        );
        loop(a1,
            if(ord(a1)-1 = flyAng(t1,thr1),
                Rdeca(t1,thr1) = redAlpha('decoy',a1)
            );
        );
    );
);
Rdec(t,thr) = Rdecd(t,thr) * Rdeca(t,thr);

loop(t1,
    loop(thr1,
        loop(d1,
            if(ord(d1)-1 = flyDist(t1,thr1),
                Rchfd(t1,thr1) = redRho('chaff',d1)
            );
        );
        loop(a1,
            if(ord(a1)-1 = flyAng(t1,thr1),
                Rchfa(t1,thr1) = redAlpha('chaff',a1)
            );
        );
    );
);
Rchf(t,thr) = Rchfd(t,thr) * Rchfa(t,thr);

///////
// Find minimum distance to threat
minDist(t1,thr1) = min(199, flyDist(t1,thr1)); // Keep
    under 200

///////
// Find the probability of a threat
loop(t1,
    loop(thr1,
```

```
            if(flyDist(t1,thr1) > 100,
                probThreat(t1,thr1) = 0;
            else
            probThreat(t1,thr1) = 1;
            );
        );
    );


//////////////////
Scalars
    bigM     Big number                              / 1000 /
    Tja      Time steps for activating the jammer    / 3 /
    Tjs      Time steps for jammer to stop           / 2 /
    Tda      Time steps for activating the decoy     / 2 /
    Tdr      Time steps for releasing the decoy      / 1 /
    Tcf      Time steps for forming the chaff cloud  / 2 /
    Tcd      Duration of the chaff cloud             / 3 /
    Tcl      Time steps for chaff latency            / 2 /
    Kd       Max number of towed decoys              / 2 /
    Kc       Max number of chaff dispensings         / 5 /
    epsilon  Penalty value                           / 0.0001 /
    ;


//////////////////
Variables
    ObjVal  The objective value
    Ssum        Sum of survivability over time
    Rmax(t,thr) Maximum reduction of threat thr to time t
    leth(t)      Lethality to time t
    Oj(t)        Jammer is turned on
    Onj(t)       Jammer gets turned on
    Offj(t)      Jammer gets turned off
    Aj(t)        Jammer is active
    Cj(t)        Count for how long the jammer has been on
    Od(t)        Decoy is deployed
    Ond(t)       Decoy gets deployed
    Offd(t)      Decoy gets released
    Ad(t)        Decoy is active
    Cd(t)        Count for how long the decoy has been deployed
    Oc(t)        Chaff is dispensed
    Ac(t)        Chaff cloud formed
    a(t,thr,cm) Countermeasure offering best reduction
        against threat thr at time t
    ;
    Binary variables Oj,Onj,Offj,Aj,Od,Ond,Offd,Ad,Oc,Ac,a;
```

```
//////////////////
Equations
    obj             Define objective function
    surv            Total survivability
    totLeth(t)      Total lethality to time t

    maxJam(t,thr)   Maximize reduction wrt. the jammer
    maxDec(t,thr)   Maximize reduction wrt. the towed decoy
    maxChf(t,thr)   Maximize reduction wrt. chaff
    maxNoCm(t,thr)  Maximize reduction wrt. the 'NoCM' dummy

    oneCm(t,thr)    Only one countermeasure for each threat
        at a time
    noCmOut(t,thr)  Set 'noCm' outside range of a threats
    noCmIn(t,thr)   Set 'noCm' inside range of a threat

    jamOn(t)        Turning the jammer on
    jamOff(t)       Turning the jammer off
    jamKeepOn(t)    Keep the jammer turned on in phase II
    jamCountMax(t)  Counting jammer on-time - max increase
    jamCountMin(t)  Counting jammer on-time - min increase
    jamCountLim(t)  Counting jammer on-time - max value
    jamCountPos(t)  Counting jammer on-time - keep positive
    jamOnLong(t)    Keep the jammer turned on in phase III
    jamOnAct(t)     Keep jammer active while on (phase III)
    jamKeepAct(t)   Keep jammer active in phase IV
    jamKeepOff(t)   Keep jammer off in phase IV

    decOn(t)        Deploying the towed decoy
    decOff(t)       Releasing the towed decoy
    decKeepOn(t)    Keep the decoy deployed in phase I
    decCountMax(t)  Counting decoy on-time - max increase
    decCountMin(t)  Counting decoy on-time - min increase
    decCountLim(t)  Counting decoy on-time - max value
    decCountPos(t)  Counting decoy on-time - keep positive
    decOnAct(t)     Keep decoy active while deployed (phase
        III)
    decKeepOnAct(t) Keep decoy turned on in phase III
    decKeepOff(t)   Keep decoy off in phase IV
    decMaxDepl      Maximum number of decoy deployments

    chfCloudForm(t) The chaff cloud is formed
    chfCloudDiss(t) The chaff cloud is dissolved
    chfLatency(t)   Latency between chaff dispensings
    chfMaxDisp      Maximum number of chaff dispensings
```

```
    ;

alias(t,j);

// Objective function
obj              ..  ObjVal   =e= Ssum - sum(t, epsilon*
    (Oj(t)+Onj(t)+Offj(t)+Aj(t)+Aj(t)+
    Od(t)+Ond(t)+Offd(t)+Ad(t)+Oc(t)+Ac(t)));
surv             ..  Ssum     =e= sum(t, 1-leth(t)) / card(t);
totLeth(t)       ..  leth(t)  =e= sum(thr,
    probThreat(t,thr)*(lethality(t,thr)/100)*(1-Rmax(t,thr)));

// General constraints
maxJam(t,thr)    ..  Rmax(t,thr)                =l= Rjmr(t,thr) *
    Aj(t) + bigM*(1-a(t,thr,'jammer'));
maxDec(t,thr)    ..  Rmax(t,thr)                =l= Rdec(t,thr) *
    Ad(t) + bigM*(1-a(t,thr,'decoy'));
maxChf(t,thr)    ..  Rmax(t,thr)                =l= Rchf(t,thr) *
    Ac(t) + bigM*(1-a(t,thr,'chaff'));
maxNoCm(t,thr)   ..  Rmax(t,thr)                =l=
    bigM*(1-a(t,thr,'noCm'));

oneCm(t,thr)     ..  sum(cm, a(t,thr,cm))       =e= 1;
noCmOut(t,thr)   ..  a(t,thr,'noCm')            =l=
    flyDist(t,thr)/100;
noCmIn(t,thr)    ..  a(t,thr,'noCm')            =g=
    (minDist(t,thr)/100)-1;

// Jammer constraints
jamOn(t)         ..  Onj(t)                     =g= Aj(t+Tja) -
    Aj(t+(Tja-1));
jamOff(t)        ..  Offj(t)                    =g= Oj(t-1) -
    Oj(t);
jamKeepOn(t)     ..  sum(j$(ord(j) >= ord(t) and ord(j) <=
    ord(t)+Tja-1), Oj(j)) =g= Tja*Onj(t);
jamCountMax(t)   ..  Cj(t)                      =l= Cj(t-1) + 1;
jamCountMin(t)   ..  Cj(t) + card(t)*(1 - Oj(t)) =g= Cj(t-1) +
    1;
jamCountLim(t)   ..  Cj(t) - card(t) * Oj(t)    =l= 0;
jamCountPos(t)   ..  Cj(t)                      =g= 0;
jamOnLong(t)     ..  Aj(t+Tjs)                  =l= 1-Offj(t);
jamOnAct(t)      ..  card(t) * Aj(t)            =g= Cj(t) - Tja;
jamKeepAct(t)    ..  sum(j$(ord(j) >= ord(t) and ord(j) <=
    ord(t)+Tjs-1), Aj(j)) =g= Tjs * Offj(t);
jamKeepOff(t)    ..  sum(j$(ord(j) >= ord(t) and ord(j) <=
    ord(t)+Tjs-1), Oj(j)) =l= Tjs * (1-Offj(t));
```

```
// Decoy constraints
decOn(t)          ..  Ond(t)                        =g= Ad(t+Tda) -
    Ad(t+(Tda-1));
decOff(t)         ..  Offd(t)                       =g= Ad(t-1) -
    Ad(t);
decKeepOn(t)      ..  sum(j$(ord(j) >= ord(t) and ord(j) <=
    ord(t)+Tda-1), Od(j)) =g= Tda * Ond(t);
decKeepOnAct(t)   ..  Od(t)                         =g= Ad(t);
decCountMax(t)    ..  Cd(t)                         =l= Cd(t-1) + 1;
decCountMin(t)    ..  Cd(t) + card(t)*(1 - Od(t)) =g= Cd(t-1) +
    1;
decCountLim(t)    ..  Cd(t) - card(t)* Od(t)  =l= 0;
decCountPos(t)    ..  Cd(t)                         =g= 0;
decOnAct(t)       ..  card(t) * Ad(t)               =g= Cd(t) - Tda;
decKeepOff(t)     ..  sum(j$(ord(j) >= ord(t) and ord(j) <=
    ord(t)+Tdr-1), Od(j)) =l= Tdr * (1-Offd(t));
decMaxDepl        ..  sum(t, Ond(t))                =l= Kd;

// Chaff constraints
chfCloudForm(t)  ..  sum(j$(ord(j) >= ord(t)+Tcf and ord(j) <=
    ord(t)+Tcf+Tcd-1), Ac(j)) =g= Tcd * Oc(t);
chfCloudDiss(t)  ..  Ac(t)                         =l= sum(j$(ord(j)
   >= ord(t)-Tcd-Tcf+1 and ord(j) <= ord(t)-Tcf), Oc(j));
chfLatency(t)     ..  sum(j$(ord(j) >= ord(t)+1 and ord(j) <=
    ord(t)+Tcl), Oc(j))          =l= Tcl * (1-Oc(t));
chfMaxDisp        ..  sum(t, Oc(t))                 =l= Kc;


Model maxSurvivability /all/;
Solve maxSurvivability using mip maximizing ObjVal;


////////////////////
// Output
display Rjmr;
display Rdec;
display Rchf;
display Rmax.L;

display Onj.L;
display Offj.L;
display Oj.L;
display Aj.L;
display Cj.L;

display Ond.L;
display Offd.L;
```

```
display Od.L;
display Ad.L;

display Oc.L;
display Ac.L;
display a.L;

display Ssum.L;
display objVal.L;

File out    The results     / tempasp.res /;
out.nd=5;
out.ap=1;
Put out;
Put     system.date,
    @10 system.time,
    @20 card(t),
    @30 Ssum.L,
    @42 ObjVal.L,
    @62 system.elapsed,
    @70 '%gams.user1%';
PutClose out;
```

Appendix E

Software and Hardware

This appendix gives a short description of parts of the software and hardware involved in the development of the models described in the work.

E.1 Software

This section describes the software that is used for main parts of the development. Where extra information is available on the Internet relevant URLs are given.

B-Prolog For the development and tests of the Prolog program described in Chapter 4 the B-Prolog interpreter is used. This is chosen since it seems robust, it has an easy to use prompt interface, it is fairly well documented, and it is free for academic use.

B-Prolog can be downloaded from `http://www.probp.com/`.

HUGIN During development of the BN described in Chapter 5 version 6.3 of HUGIN is used. The tests have been performed using HUGIN™ version 6.7 (build 6702).

A limited version of HUGIN, known as HUGIN Lite™, can be downloaded for free from `http://www.hugin.com/`. Following platforms are covered: Windows, Solaris Sparc, Solaris x86, Linux, Mac OS X 10.3, and Mac OS X 10.4.

Fly-In The Fly-In 2000 GUI version 2.3.4 software is used to produce data for the Structural Learning of a BN. The software comes with the following message: "The Fly-In software was produced and released to EWS 5/SCI-067 by DSTL."

MATLAB MATLAB is used for the construction of scenarios and the sampling of time steps used in both the mathematical model (Chapter 6) and with the metaheuristics (Chapter 7). Various versions of MATLAB are used in the work. For the final work MATLAB® version 7.1.0.183, R14, Service Pack 3 is used.

See `http://www.mathworks.com/` for more information on MATLAB.

GAMS In solving the mathematical model GAMS is run on a license given to the Technical University of Denmark. The 140th revision, dated November 11th, 2004, has been used.

Information on GAMS can be found at `http://www.gams.com/`.

CPLEX GAMS uses CPLEX as solver. The name "CPLEX" comes from the combination of the letter "C" for the programming language, and the word "simplex" for the simplex method for linear programming. Version 9.130 of the ILOG CPLEX software is run. It is licensed to the Technical University of Denmark.

`http://www.ilog.com/` gives more information on the ILOG CPLEX software.

E.2 Hardware

For most work a laptop PC with an Intel Pentium™ 4 1.79 GHz Mobile CPU and 512 MB of RAM is used. The PC is running Microsoft Windows XP, and it has been used to run B-Prolog, HUGIN, Fly-In, and MATLAB.

The mathematical model was developed and run on a SUN Fire 3800 1200 MHz with 8 CPUs and 16 GB RAM running Solaris 9.

Implementing and running the metaheuristics is done using a stationary PC with an AMD Athlon™ 64 X2 Dual Core Processor 4200+ 2.21 GHz and 2 GB of RAM. It is running Microsoft Windows XP.

The dissertation is written using WinEdt/MikTeX running mainly on the laptop PC. Parts of the dissertation have been written on handheld devices such as a Palm III, a Compaq iPaq, and a Sony Ericsson K510i mobile phone.

Bibliography

[1] Directional infrared counter measures. [http://en.wikipedia.org/wiki/DIRCM], February 2007.

[2] ECL^{i}PS^e. [http://eclipse.crosscoreop.com/], March 2007.

[3] International Society for Bayesian Analysis. [http://www.bayesian.org/], March 2007.

[4] MIL-STD-1553. [http://en.wikipedia.org/wiki/MIL-STD-1553], March 2007.

[5] OODA Loop. [http://en.wikipedia.org/wiki/OODA_Loop], March 2007.

[6] The Missile Index. [http://missile.index.ne.jp/], February 2007.

[7] Stig K. Andersen, Kristian G. Olesen, Finn V. Jensen, and Frank Jensen. HUGIN – a shell for building Bayesian belief universes for expert systems. pages 332–337, 1990.

[8] Virginia Apgar. A proposal for a new method of evaluation of the newborn infant. *Current Researches in Anesthesia and Analgesia*, pages 261–262, July-August 1953.

[9] J.M. Arroyo and A.J. Conejo. Optimal response of a thermal unit to an electricity spot market. *IEEE Transactions on Power Systems*, 15(3):1098–1104, 2000.

[10] Robert E. Ball. *The Fundamentals of Aircraft Combat Survivability Analysis and Design.* AIAA Education Series. Naval Postgraduate School, 2 edition, 2003.

[11] Sheila B. Banks and Carl S. Lizza. Pilot's associate: a cooperative, knowledge-based systemapplication. *IEEE Expert*, 6(3):18–29, June 1991.

[12] Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley, 2. edition, 1990.

[13] Bruce D'Ambrosio. Inference in Bayesian networks. *AI Magazine*, 20(2):21–36, 1999.

[14] Danish Defence Research Establishment. Mååleradarsystemet mrs.

[15] Kathryn A. Dowsland. *Modern heuristic techniques for combinatorial problems*, chapter 2, pages 20–69. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[16] Morten Enevoldsen. Decision support for fighter pilots. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2003.

[17] Peter Haddawy. An overview of some recent developments in Bayesian problem solving techniques. *AI Magazine*, Summer 1999.

[18] Jonas Lundbek Hansen, Steen Søndergaard, and Erik Thisen. FOFT EK taktiske træner. *FOFT Nyt*, (3), December 2003.

[19] Hovland Harald. Optimisation of flare and chaff programs – an analytical approach. Technical Report 2006/01460, Norwegian Defence Research Establishment, 2006.

[20] Clyde W. Holsapple and Andrew B. Whinston. *Decision Support Systems – A Knowledge-Based Approach*. West Publishing Company, 1996.

[21] Finn Verner Jensen. *An introduction to Bayesian networks*. UCL Press, 1996.

[22] Finn Verner Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[23] Uffe Bro Kjærulff and Anders L. Madsen. *Probabilistic Networks – An Introduction to Bayesian Networks and Influence Diagrams*. Aalborg University, 2005.

[24] E. C. Labatt, Jr., editor. *Automated threat response recommendation in environments of high data uncertainty using the Countermeasure Association Technique (CMAT)*, September 1991.

[25] Helge Langseth and Thomas D. Nielsen. Fusion of domain knowledge with data for structural learning in object oriented domains. *Journal of Machine Learning Research*, 4, 2003.

[26] Steffen L. Lauritzen. The EM algorithm for graphical association models with missing data. *Comput. Stat. Data Anal.*, 19(2):191–201, 1995.

[27] H.R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to iterated local search. In *Proceedings of the Fourth Metaheuristics International Conference*, volume 1, pages 1–6, 2001.

[28] Anders L. Madsen, Michael Lang, Uffe B. Kjærulff, and Frank Jensen. The Hugin tool for learning Bayesian networks. In *Lecture Notes in Computer Science*, volume 2711, April 2004.

[29] Bruce A. McCarl. *GAMS User Guide*. Texas A&M University, 22.2 edition, March 2006. Developed in cooperation with GAMS Development Corporation.

[30] Jamison Jo Medby and Russell W. Glenn. *Street Smart: Intelligence Preparation of the Battlefield for Urban Operations*. 2002.

[31] F. W. Moore. A methodology for missile countermeasures optimization under uncertainty. *Evolutionary Computation*, 10(2):129–149, 2002.

[32] F. W. Moore and O. N. Garcia. A genetic programming methodology for missile countermeasures optimization under uncertainty. *Lecture Notes in Computer Science*, (1447):367–376, 1998.

[33] John H. Painter, III Wallace E. Kelly, Jeffrey A Tang, Kristopher A. Lee, Paul A. Branham, John W. Crump, Donald T. Ward, Karthik Krishnamurty, Disk L. Y. Woo, William P. Alcorn, Andrew C. Robbins, and Ren-Jye Yu. Decision support for the general aviation pilot. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, October 1997.

[34] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[35] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in FORTRAN: the art of scientific computing*. Cambridge University Press, New York, NY, USA, 2. edition, 1992.

[36] A. Quan, R. Crawford, H. Shao, K. Knudtzon, A. Schuler, D. Scott, S Hayati, Jr Higginbotham, R., and R. Abbott. Automated threat response using intelligent agents (ATRIA). *Aerospace Conference, 2001, IEEE Proceedings*, 6:2721–2730, March 2001.

[37] Per Husmann Rasmussen. *Feasibility-studie, Intelligente Systemer*, chapter 4, „Beslutningsstøttesystem", pages 15–19. Royal Danish Defence College, March 2003.

[38] Mathias Ravn and Lars Vadstrup Hansen. Neuroevolution af computerstyrede pokerspillere uden anvendelse af ekspertstrategi. Master's thesis, University of Aarhus, November 2005.

[39] William B. Rouse, Normann D. Geddes, and John M. Hammer. Computer-aided fighter pilots. *IEEE Spectrum*, 27(3):38–41, March 1990.

[40] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1999.

[41] D. Curtis Schleher. *Introduction to Electronic Warfare*. Artech House, 1986.

[42] M.I. Skolnik. *Radar Handbook*. McGraw-Hill, 1970.

[43] Steen Søndergaard. FOFT støtter flyvevåbnet ved NATO forsøg. *FOFT Nyt*, (2), August 1998.

[44] P. Spirtes, C. Glymour, and R. Schneies. *Causation, Prediction, and Search. Adaptive Computation and Machine Learning*. MIT Press, 2 edition, January 2000.

[45] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1), 1991.

[46] G.W. Stimson. *Introduction to Airborne Radar*. SciTech Publishing, Inc., 2. edition, 1998.

[47] Dan Strömberg. Decision-making using temporal reasoning. *IJCAI'99 Workshop on Teams*, 1999.

[48] Peter Svenmarck. Decision support in a fighter aircraft: From expert systems to cognitive modelling. HFA Report 1998-04, Linköpings Universitet, Swedish Centre for Human Factors in Aviation, 1998.

[49] Peter Svenmarck and Sidney Dekker. Decision support in fighter aircraft: From expert systems to cognitive modelling. *Behaviour and Information Technology*, 22(3):175–184, 2003.

[50] G. Teasdale and B. Jennett. *LANCET (ii)*, pages 81–83, 1974.

[51] Terma. AN/ALQ-213(V) Electronic Warfare Management System. Brochure.

[52] Jim D. Titley. Integrated Defensive Aids Systems, a matter of definition. DDRE Report F-13/2003, Danish Defence Research Establishment, 2003.

[53] Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the eighth conference on Uncertainty in Artificial Intelligence*, pages 323–330, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[54] Patrick Henry Winston. *Artificial Intelligence*. Series in Computer Science. Addison-Wesley, 2. edition, 1984.

[55] Fred Wright. Automated electronic warfare with threat response processing. Georgia Tech Research Institute, May 2006.

[56] Neng-Fa Zhou. *B-Prolog User's Manual*. Afany Software, December 2006.